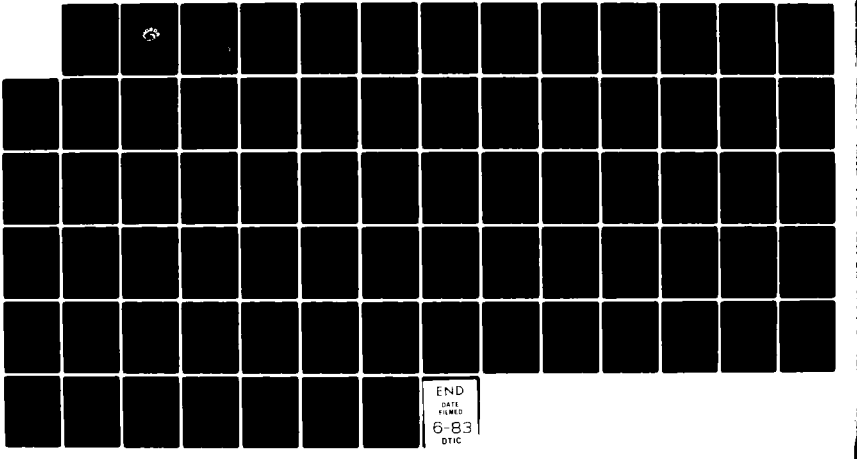
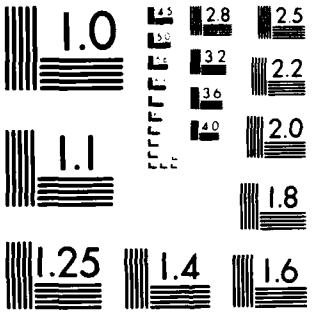
AD-A127 650 SYNTACTIC/SEMANTIC TECHNIQUES FOR FEATURE DESCRIPTION 1/1
AND CHARACTER RECOGNITION(U) PERCEPTICS CORP KNOXVILLE
TN R C GONZALEZ JAN 83 NORDA-TN-185 N00014-80-M-0030
UNCLASSIFIED F/G 6/4 NL

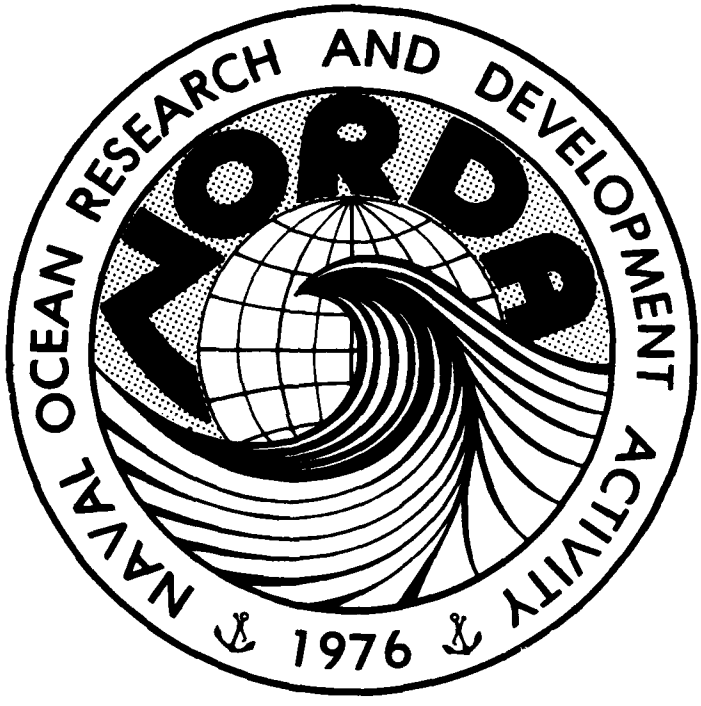




NORDA Technical Note 185

Naval Ocean Research
and Development Activity
NSTL Station, Mississippi 39529

Syntactic/Semantic Techniques for Feature Description and Character Recognition



Prepared by NORDA Code 370, Mapping, Charting and Geodesy Division for NORDA Code 550, Mapping, Charting and Geodesy Program Management Office, Ocean Science and Technology Laboratory

Dr. R.C. Gonzalez
Perceptics Corporation
221B Clark Street
Knoxville, TN 37921

Sponsored by Defense Mapping Agency, HQ STT

January 1983

PREFACE

The Pattern Analysis Branch, Mapping, Charting and Geodesy (MC&G) Division, of the Naval Ocean Research and Development Activity (NORDA) has been involved over the past several years in the development of algorithms and techniques for computer recognition of free-form handprinted symbols as they appear on the Defense Mapping Agency (DMA) maps and charts. NORDA has made significant contributions to the automation of MC&G through advancing the state of the art in such information extraction techniques. In particular, new concepts in character (symbol) skeletonization, rugged feature measurements, and expert system-oriented decision logic have allowed the development of a very high performance Handprinted Symbol Recognition (HSR) system for identifying depth soundings from naval smooth sheets (accuracies greater than 99.5%).

The study reported in this technical note is part of NORDA's continuing research and development in pattern and shape analysis as it applies to Navy and DMA ocean/environment problems. The issue addressed in this technical note deals with emerging areas of syntactic and semantic techniques in pattern recognition as they might apply to the free-form symbol problem. The author was asked to review these powerful tools in light of his earlier support to the Pattern Analysis Laboratory [1] and to analyze their potential for extending the HSR system to a wider range of symbols. These results contribute to the overall NORDA R&D effort to investigate and develop methods for more precise geometric shape descriptions for application to Ocean Science Information Extraction (OSIS) problems.

ACKNOWLEDGMENTS

This work was sponsored by DMA under Program Element 63701B, under the Subtask entitled "Optical Character Recognition and was performed for NORDA Code 550, the Mapping, Charting, and Geodesy Program Management Office. The DMA Program Manager was Robert Penny, and LCDR Vic Hultstrand (Code 550) was the project manager. Dr. Robert M. Brown (Code 371) was the scientific officer. Dr. Ralph Gonzalez of Perceptics Corporation was under contract to NORDA Code 370 for the preparation of this study.

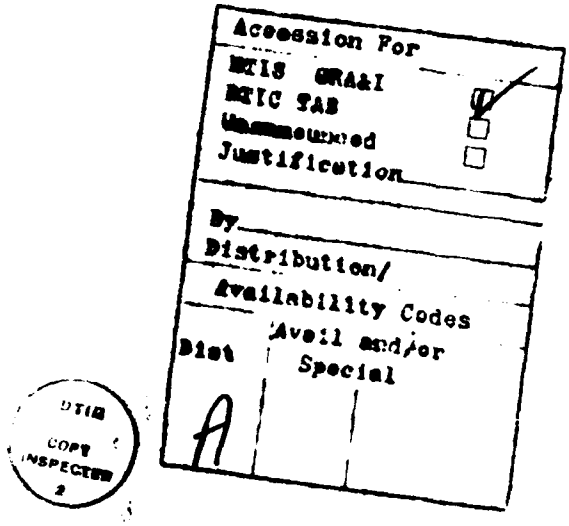


TABLE OF CONTENTS

I. INTRODUCTION

This report discusses a character recognition approach based on the use of syntactic/semantic concepts. This approach is consistent with our earlier work for NORDA in the sense that it is based on the same philosophy and types of features that were recommended in an earlier report [1] and which have been investigated since then. The material in the following sections unifies these recommendations and includes extensions such as techniques for handling interconnections between features, the recognition of feature strings by syntactic methods, and the use of semantics for quality assurance both in the computation of features and in the recognition stage. The methods described in this report are intended as a complement to the techniques presently being used in the NORDA OCR system, and as a potential tool for handling forthcoming problems in alphanumeric character recognition.

The structure of the proposed approach is shown diagramatically in Fig. 1. It is assumed that the input to the system is a skeleton of the character to be recognized. The selection of a skeleton input is consistent with the processing capabilities of the present OCR system. A skeleton representation also has the advantage that it facilitates the computation of features such as bays, lakes, and branch points, which have been deemed essential for rugged character representations. It is noted, however, that the methods discussed in the following sections could easily be modified to accept character outline (border) inputs.

The feature extraction and attribute assignment stage has the function of computing and quantifying all the features required for recognition. As explained in more detail in Section 2, this stage is based on a hierarchical, semantic-guided approach. The function of the screening stage is to select a particular set of recognition modules to process a given input. The basic idea is that, at this point in the overall process, the features detected in a character can be used to advantage in guiding the recognition strategy to be applied to the input. The pre-selection of a subset of recognition procedures not only simplifies

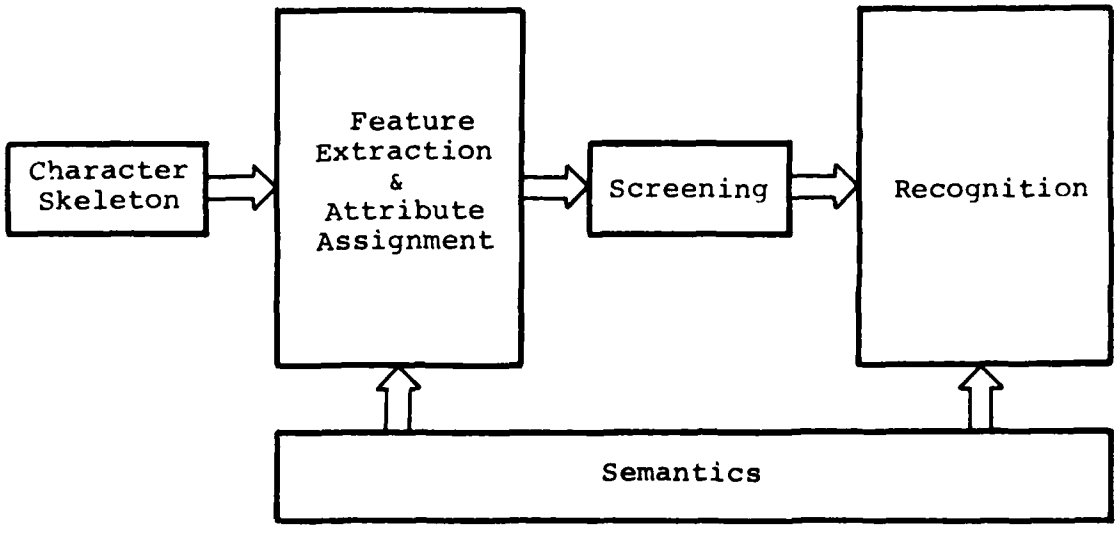


Figure 1. Syntactic/semantic character processor.

the organization of the classifier, but has the added advantage of operational efficiency. The recognizer is based on the use of syntactic/semantic techniques. As described in Section 3, the syntax establishes the structure of the pattern classes under consideration, while the semantics establish the meaning or validity of a particular pattern in the context of that structure.

One of the most important aspects in the selection of any pattern recognition approach is the availability of learning algorithms. The techniques discussed in this report are formulated to take advantage of one of the most powerful learning techniques available for syntactic systems. As discussed in Section 4, the proposed learning algorithm depends on only one user-specified parameter, and the behavior of the procedure as a function of this parameter is well understood and easily analyzed.

Another important advantage of the approach discussed in the following sections is that it includes a procedure for checking class separability. Given the recognizers learned from a set of training pattern classes, the procedure discussed in Section 5 identifies the patterns that cannot be classified into a unique class, thus yielding information related to the discriminatory power of the features used in the system, and the structure of the patterns in the overlapping regions.

Although as indicated above, learning algorithms already exist for the syntactic components of the character processor, no such algorithms are yet known for the semantics. The material in Section 6 addresses this problem from an interactive point of view which utilizes automatic learning for the syntax and user-defined rules to establish the corresponding semantic components of the system.

II. FEATURE EXTRACTION AND ATTRIBUTE ASSIGNMENT

2.1 Background

In this section we discuss a hierarchical, semantics-based approach for feature extraction, as well as the assignment of attributes to those features.

The basic approach is shown diagramatically in Fig. 2. For features at level k of the hierarchy, we consider a structural description of the form

Level k/features/attributes/primitives/semantic rules

The hierarchical nature of the method implies that the procedure starts with simple primitives and successively builds more complex features from them. It is noted that what we call primitives in the computation of a feature at level k may be features that have been computed at levels lower than k. This terminology is used for consistency in the structural description given above.

The attributes are used for characterizing each feature with descriptors such as length, orientation, and location of its centroid. The use of semantics allows quality control of the features generated at all levels of the hierarchy. The approach is to use semantic rules in order to guarantee that all features used for subsequent recognition are meaningful in the context of character recognition.

Level zero of the hierarchy consists of the input data to the recognition system (e.g., character skeletons). The function of the other levels is explained in the following sections.

2.2 Level One: End Points

Level 1 of the hierarchy extracts all end points in a given skeleton. The structural description is

Level 1/end point/attributes/primitives/semantic rules

where the elements of the description are as follows:

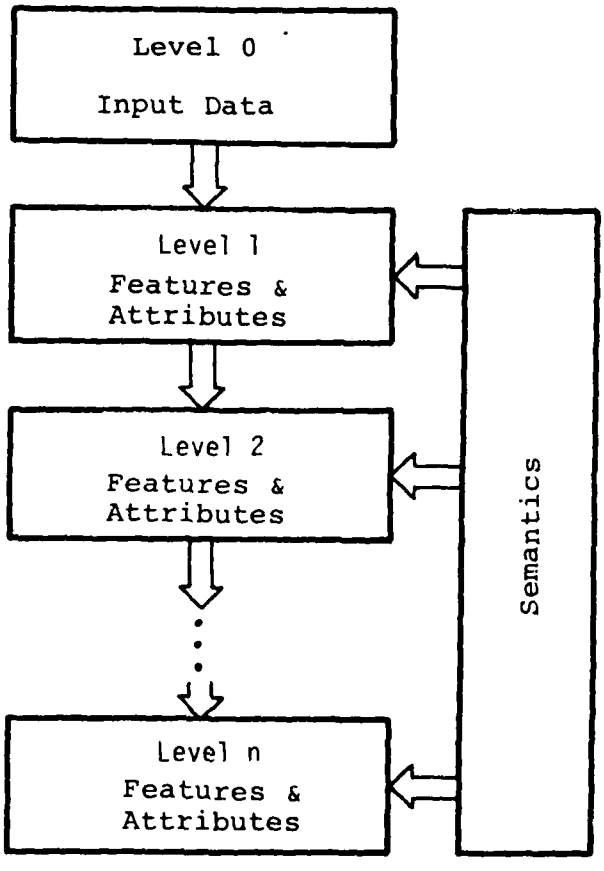


Figure 2. Hierarchical, semantic-guided feature extraction and attribute assignment.

Attributes

The only attribute assigned to an end point is its location.

Primitives

The primitive of an end point is a single pixel satisfying the following rule:

Semantic Rule

$R_1(1)$ = TRUE for any pixel with exactly one m-neighbor.

Thus, all pixels for which $R_1(1)$ is TRUE are labeled as end-point features.

2.3 Level Two: Branch Points

Level 2 of the hierarchy extracts branch points using the following description

Level 2/branch point/attributes/primitives/semantic rules

where the elements of the description are as follows:

Attributes

The attributes assigned to a branch point are its location and number of branches attached to it.

Primitives

The only primitive of a branch point is a single pixel satisfying the following rule:

Semantic rule

$R_1(2)$ = TRUE for any pixel with more than two, and less than $T_1(2)$, m-neighbors, where $T_1(2)$ is a threshold (e.g., four).

All pixels in an input skeleton for which $R_1(2)$ is TRUE are labeled as branch-point features.

2.4 Level Three: Arcs

Arc features have the structural description

Level 4/arc/attributes/primitives/semantic rules

where the elements of the description are as follows:

Attributes

The attributes assigned to an arc are the location of its two terminator points and its length. A terminator point in this case is either an end point or a branch point,† and the arc length is the checker-board distance between the terminator points.

Primitives

The primitives of an arc are the set of pixels satisfying the following semantic rules:

Semantic Rules

$R_1(3)$ = TRUE if only two distinct pixels are terminator points.

$R_2(3)$ = TRUE if there is only one set of pixels, each pixel having exactly two m-neighbors, and lying between the terminator points identified in $R_1(3)$.

An arc feature is then a set of pixels for which $R_1(3) \cap R_2(3)$ = TRUE.

2.5 Level Four: Lakes

The structural description for lakes has the form

Level 4/lake/attributes/primitives/semantic rules

Attributes

The attributes assigned to a lake feature are: (1) the location of its centroid: (2) the error of its least-square-error elliptical fit (see Appendix A); (3) the direction of its principal axes, $d_1(4)$ and $d_2(4)$; (4) the variance (spread) along each principal axis, $v_1(4)$ and $v_2(4)$; and (5) the location of any branch points along the boundary.

Primitives

The primitives of a lake are pixels satisfying the following semantic rules:

---

†More generally, a terminator point is any pixel that denotes the end of a feature. The feature may be embedded between two other features, in which case both terminator points could be internal pixels.

Semantic rules

$R_1(4)$ = TRUE if there is a set of m-connected pixels (including branch points) forming a closed boundary.

$R_2(4)$ = TRUE if the elliptical fit error is less than a threshold $T_1(4)$.

$R_3(4)$ = TRUE if the ratio $v_1(1)/v_2(1)$ is less than a threshold $T_2(4)$, where it is assumed that $v_1(1) \geq v_2(1)$.

Thus, a lake feature is a set of pixels for which $R_1(4) \cap R_2(4) \cap R_3(4)$ = TRUE. It is noted that rule $R_1(4)$ establishes a lake in the general sense that it refers to a closed boundary. Rules $R_2(4)$ and $R_3(4)$, however, further refine this concept by establishing a valid lake shape for the purpose of character recognition.

2.6 Level Five: Polygonal Segments

The features discussed in this and the following three sections deal with the characterization of arcs. The first step is to approximate a given arc by a set of connected polygonal segments using the structural description

Level 5/polygonal segment/attributes/primitives/semantic rules

The elements of this description are as follows:

Attributes

The attributes used for each polygonal segment are: (1) length, (2) direction, (3) location of terminator points, (4) location of centroid (i.e., midpoint), and (5) approximation error.

Primitives

The primitives of polygonal segments are the pixels in a given arc.

Semantic rules

$R_1(5)$ = TRUE if the mean-squared error between a polygonal segment and its corresponding arc is less than a threshold $T_1(5)$.

$R_2(5)$ = TRUE if the number of polygonal segments satisfying $R_1(5)$ is less than a threshold $T_2(5)$.

We say that a polygonal approximation of a given arc is valid if $R_1(5) \cap R_2(5)$ = TRUE. Semantic rule $R_1(5)$ establishes, by means of $T_1(5)$, an acceptable approximation in a mean-squared-error sense. Since it is always possible to make all errors arbitrarily small (the limiting case is zero by using n - 1 segments, where n is the number of pixels), semantic rule $R_2(5)$ is used as an "irregularity filter." That is, preselecting the maximum number of polygonal segments that are allowed eliminates as unacceptable irregular arcs that require a greater number of segments in order to satisfy the error criterion in Rule $R_1(5)$.

The direction attribute of each polygonal segment is quantized into one of eight possible directions, as shown in Fig. 3. Since two directions differing by 180^0 are possible for each segment, the ambiguity is resolved by assuming a standard clockwise, up-down scan of the polygonal structure.

2.7 Level Six: Straight-Line Segments

At this level in the hierarchy we consider straight-line segments (SLS's) which are the least complex features that can be formed using polygonal segments as primitives. The structural description is as follows:

Level 6/SLS/attributes/primitives/semantic rules

Attributes

The attributes of each SLS are: (1) length, (2) direction, (3) location of its centroid, and (4) location of its terminator points. Although, as will be seen below, an SLS may be composed of a series of polygonal segments, the length of an SLS feature is defined as the Euclidean distance between its terminator points, its direction is defined as the direction of a line passing through these two points, and its centroid is simply the midpoint. The direction attribute is encoded using the approach indicated in the previous section.

Primitives

The primitives of an SLS are polygonal segments satisfying the following semantic rules:

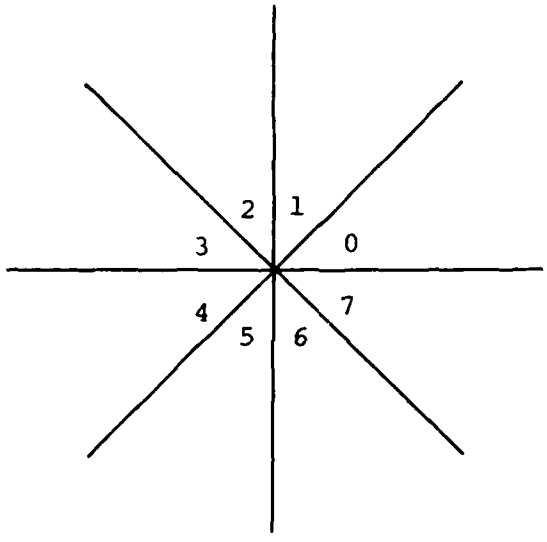


Figure 3. Octants used for quantizing direction.

Semantic Rules

$R_1(6)$ = TRUE if two contiguous polygonal segments have an interior angle greater than a threshold $T_1(6)$.

$R_2(6)$ = TRUE if a single polygonal segment has length greater than $T_2(6)$*L where L is the sum of the lengths of all the polygonal segments and $T_2(6)$ is a constant less than one.

$R_3(6)$ = TRUE if there is only one polygonal segment.

Rule $R_1(6)$ is applied recursively and any segments for which $R_1(6) \cup R_2(6) \cup R_3(6)$ = TRUE are classified as SLS's.

2.8 Level Seven: Corners

Corners† have the structural description

Level 7/corner/attributes/primitives/semantic rules

Attributes

The attributes are: (1) angle, (2) depth, (3) width, (4) area, (5) direction, (6) length of sides, (7) location of the terminator points, (8) location of the centroid, (9) convexity, and (10) concavity. The meaning of these attributes may be explained with the aid of Fig. 4. The angle of a corner is defined to be the interior angle formed by the two sides. If we treat the corner as a triangle, as shown in Fig. 4, the width is defined as the length of the base of the triangle, while the depth is the length of its altitude. The area is the area of the triangle. The direction of the corner (quantized as in Fig. 3) is given by the direction of the altitude segment directed from the corner point to the base. The centroid of a corner is the average of the centroids of its sides. To establish whether a corner is convex or concave we consider a traveler traversing the corner in a clockwise up-down manner. If the base of the triangle lies to the travelers right hand, the corner is convex; otherwise it is concave.

---

†This classification of corners is not related to the measures of cornericity discussed in Appendix B.

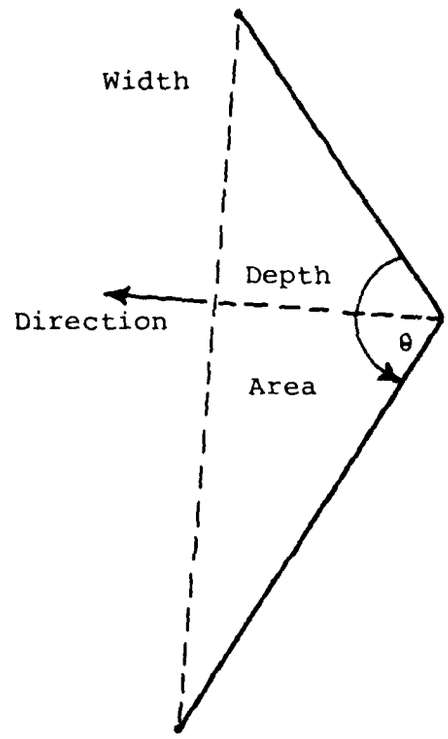


Figure 4. Attributes for corners.

Primitives

The primitives of corners are SLS's satisfying the following semantic rules:

Semantic rules

$R_1(7)$ = TRUE if $R_3(6)$ = FALSE

$R_2(7)$ = TRUE if there remain two or more contiguous SLS's after the recursive application of rule $R_1(6)$.

If $R_1(7) \cap R_2(7)$ = *TRUE we have the condition for at least one corner.* If there are more than two contiguous SLS's, they are considered pairwise, each contiguous pair forming a corner and possibly sharing sides with other contiguous corners.

2.9 Level Eight: Bays

The final features computed by the hierarchical feature extractor are bays, which have the structural description

Level 8/bay/attributes/primitives/semantic rules

Attributes

The attributes of a bay are: (1) opening, (2) area, (3) direction (4) degree, (5) height-to-width ratio (H/W), (6) length, (7) location of the terminator points, (8) location of the centroid, (9) convexity, and (10) concavity. The opening attribute is simply the Euclidean distance between the two terminator points. The area is the sum of the areas of the corners forming the bay (see below). The direction of a bay is the average of the directions of the corners. The degree of a bay is defined as the number of corners of which it is composed. The attribute H/W is the height-to-width ratio of a bounding box (the orientation of the box could be along the principal eigen axes or simply along the x-y extremes). The length of a bay is equal to the sum of the lengths of the sides of the corners. The centroid is the average of the corner centroids. Finally, a convex (concave) bay is formed by two or more convex (concave) corners.

Primitives

The primitives of a bay are corners satisfying the following semantic rule:

Semantic rule

$R_1(8)$ = TRUE if there are two or more convex (concave) contiguous corners.

Thus, the presence of a bay is established simply by combining corner features which are contiguous and have the same convexity or concavity attribute.

The material in Section 2.2 through 2.9 is summarized in Table 1 and illustrated in the following example.

Example

The concepts discussed thus far are illustrated in Fig. 5. The input to the hierarchical feature extractor is shown in Fig. 5(a), and the result of the first level of processing is to attach the labels E_1 and E_2 to the two end points in the character, as shown in Fig. 5(b). Level 2 produces a null output (there are no branch points), while Level 3 identifies arc_1 and terminator points t_1 and t_2. It is noted that t_1 and t_2 are simply the end points found in Level 1. Since there are no lakes in the character, Level 4 produces a null output. Level 5 produces polygonal segments s_1 through s_5, as shown in Fig. 5(d). Note the introduction of terminator points t_3 through t_6 used to denote the ends of the polygonal segments. Level 6 has the output shown in Fig. 5(e). Polygonal segments s_3 and s_4 were combined in this case into straight-line segment SLS_3 and consequently, terminator point t_5 is no longer of interest. Level 7 produces corners c_1, c_2, and c_3, along with their terminator points. It is noted that t_4 is a terminator point for both c_1 and c_3 and that c_2 has terminator points t_3 and t_6. Finally, the output of Level 7 is shown in Fig. 5(g). It combined corners c_1 and c_2 into bay_1 with terminator points t_1 and t_6. Thus, the highest-level description of the input character consists of bay_1 followed by $corner_3$.

It is important to note that all the information computed at a given level is available to all higher levels. Thus, the descriptors associated

TABLE 1

ELEMENTS OF THE HIERARCHICAL FEATURE EXTRACTOR

Level	Features	Attributes	Primitives	Semantic Rules	Condition for Validity
1	End points	Location	Pixel	$R_1(1)$ = TRUE for any pixel with exactly one m-neighbor.	$R_1(1)$ = TRUE
2	Branch points	1) Location 2) No. of branches	Pixel	$R_1(2)$ = TRUE for any pixel with more than two, and less than $T_1(2)$, m-neighbors.	$R_1(2)$ = TRUE
3	Arcs	1) Location of terminator points 2) Length	Pixels	$R_1(3)$ = TRUE if only two distinct pixels are terminator points. $R_2(3)$ = TRUE if there is only one set of pixels, each pixel having exactly two m-neighbors, and lying between the terminator points identified in $R_1(3)$.	$R_1(3) \cap R_2(3)$ = TRUE
4	Lakes	1) Location of centroid 2) Error of elliptical fit 3) Direction of the two principal axes 4) Variance along each axis 5) Location of any branch point along the boundary	Pixels	$R_1(4)$ = TRUE if there is a set of m-connected pixels (including branch points) forming a closed boundary. $R_2(4)$ = TRUE if the elliptical-fit error is less than $T_1(4)$. $R_3(4)$ = TRUE if the ratio of the variances is less than $T_2(4)$.	$R_1(4) \cap R_2(4) \cap R_3(4)$ = TRUE
5	Polygonal segments	1) Length 2) Direction 3) Location of terminator points 4) Location of centroid 5) Approximation error	Pixels in a given arc	$R_1(5)$ = TRUE if the approximation error is less than $T_1(5)$. $R_2(5)$ = TRUE if the number of polygonal segments satisfying $R_1(5)$ is less than $T_2(5)$.	$R_1(5) \cap R_2(5)$ = TRUE
6	Straight-line segments (SLS's)	1) Length 2) Direction 3) Location of terminator points 4) Location of centroid	Polygonal segments	$R_1(6)$ = TRUE if two contiguous polygonal segments have an interior angle greater than $T_1(6)$. $R_2(6)$ = TRUE if a single polygonal segment has length greater than $T_2(6)*L$, where L is the sum of the lengths of all the polygonal segments and $T_2(6) < 1$. $R_3(6)$ = TRUE if there is only one polygonal segment.	$R_1(6) \cup R_2(6) \cup R_3(6)$ = TRUE

TABLE 1

(Continued)

Level	Features	Attributes	Primitives	Semantic Rules	Condition for Validity
7	Corners	1) Angle 2) Depth 3) Width 4) Area 5) Direction 6) Length of sides 7) Location of terminator points 8) Location of centroid 9) Convexity 10) Concavity	SLS's	$R_1(7)$ = TRUE if $R_2(6)$ = FALSE $R_2(7)$ = TRUE if there remain two or more contiguous SLS's after the recursive application of rule $R_1(6)$.	$R_1(7) \cap R_2(7)$ = TRUE
8	Bays	1) Opening 2) Area 3) Direction 4) Degree 5) H/W 6) Length 7) Location of terminator 8) Location of centroid 9) Convexity 10) Concavity	Corners	$R_1(8)$ = TRUE if there are two or more convex (concave) corners.	$R_1(8)$ = TRUE

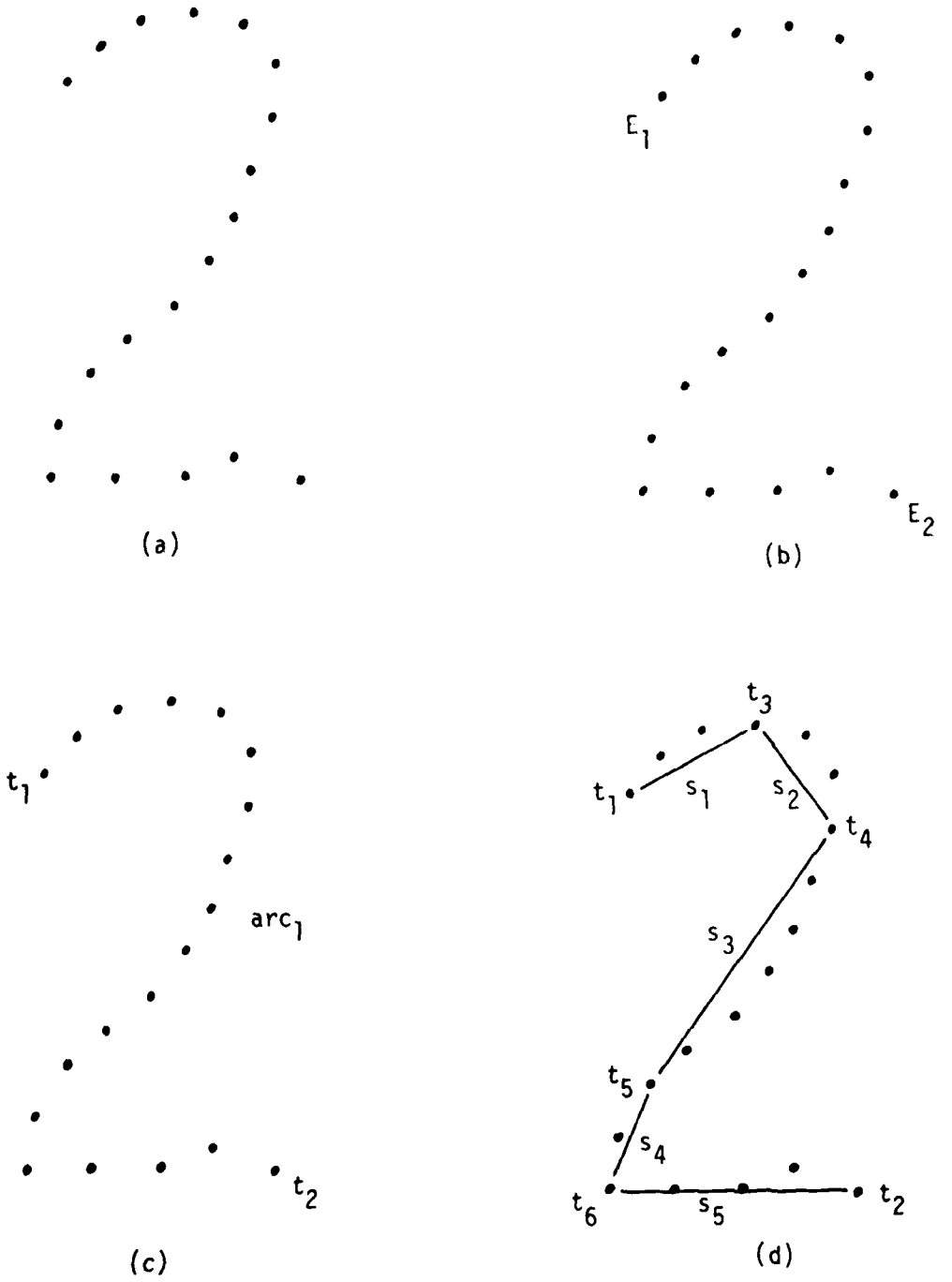
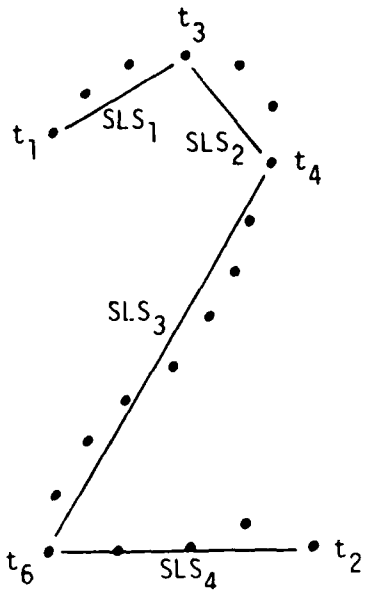
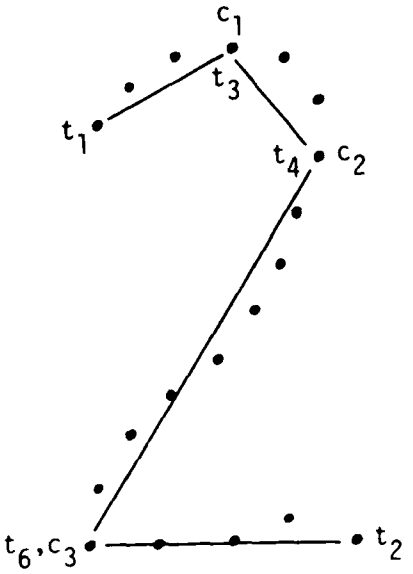


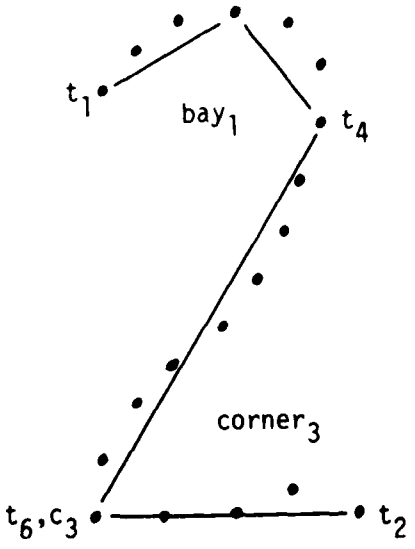
Figure 5. Illustration of the hierarchical feature extractor.



(e)



(f)



(g)

Figure 5. (Continued).

with the location of terminator points, length and direction of line segments, etc., are implicitly available to further refine the final features for the purpose of establishing their semantic validity.

2.10 Coding of Feature Strings

Once individual features have been extracted and their attributes computed, the next processing step is to organize the features in the form of a string suitable for the syntactic/semantic recognizer. A set of m features extracted from a given input character will be represented by the string notation

$$\alpha = f_1 f_2 \ldots f_m$$

where each f_i represents any of the features obtained by the procedures discussed in Sections 2.2 through 2.9.

As discussed in Section 3, the basic structure of a character will be inherent in its string representation, while semantic rules will be used for quantitatively refining the information available in a given string. In order to simplify the notation, the following codes will be used to denote the features discussed in the preceding sections.

p: branch point
a: lake
s: straight line segment
c: corner
b: bay
x: convex
v: concave

It is noted that the last two codes refer to attributes which apply to corners and bays. Although these two attributes could be incorporated in the semantic rules, they are included in the string representation as a rugged, overall descriptor to differentiate, for example, between 5's and S's. The refinement of a given corner or bay (i.e., direction, length, etc.) will be handled via the semantics. As indicated in Section 3, the degree of information incorporated in a string vs. a semantic specification is an arbitrary trade-off. The approach taken here

is, to include in a string, features and descriptors which can aid in making gross decisions between characters early in the recognition stage. It is also noted that end points, arcs, and polygonal segments are not included in the feature codes. The reason for this is that they are either implicit in, or refined into, one or more of the features coded above prior to recognition.

In order to reduce the complexity of the recognition stage, it is advantageous to organize all coded strings in a systematic manner. One way to accomplish this is to group all features in order of increasing complexity in the feature hierarchy discussed in Sections 2.2 through 2.9. For example, a character composed of a bay, a lake, a branch point, and a straight-line segment would be coded as the string α = pasb. The descriptors x and v precede the feature they modify. If the bay in this example were convex, the complete string representation would then be α = pascb. Multiple features of the same class are grouped together. If, for instance, there were two branch points the string would be α = ppascb.

III. SYNTACTIC/SEMANTIC RECOGNITION

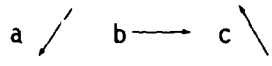
3.1 Background

The existence of recognizable, finitely describable structure in a pattern is essential for success in the syntactic approach. Basically, a formal grammar is developed to generate elements of a language that defines a pattern class, and an automaton or a parsing algorithm is used to recognize precisely that language [2,3].

For example, the grammar $G_1 = (N,\Sigma,P,S,)$ with nonterminals $N = \{S,B,C\}$, terminals $\Sigma = \{a,b,c\}$, productions $P = \{S \rightarrow aSBC,\ S \rightarrow aBC,\ CB \rightarrow BC,\ aB \rightarrow ab,\ bB \rightarrow bb,\ bC \rightarrow bc,\ cC \rightarrow cc\}$, and starting symbol S, can be shown to generate the language

$$L(G_1) = \{y | y = a^n b^n c^n,\ n \geq 1\} .$$

If the terminals a,b,c have an interpretation as pattern primitives which are unit-length directed line segments



then $L(G_1)$ defines a class of equilateral triangles via a trace of the boundary of a triangle, as shown in Fig. 6. The nature of the productions makes grammar G_1 a context-sensitive grammar.

Many, if not all, practical pattern recognition systems that use structural models are in fact hybrid systems [4]; that is, they are combinations of structural modeling techniques (primarily using syntactic models) and classical decision-theoretic techniques. One frequently finds that decision-theoretic methods are used to identify and extract the primitives in a given pattern, then syntactic methods are used to attempt the final classification by an analysis of the relationships among the primitives.

The productions in a grammar like G_1 above are purely syntax rules. They define implicitly the form that strings of terminals must have in order to belong to the language $L(G_1)$. In the example given above, that

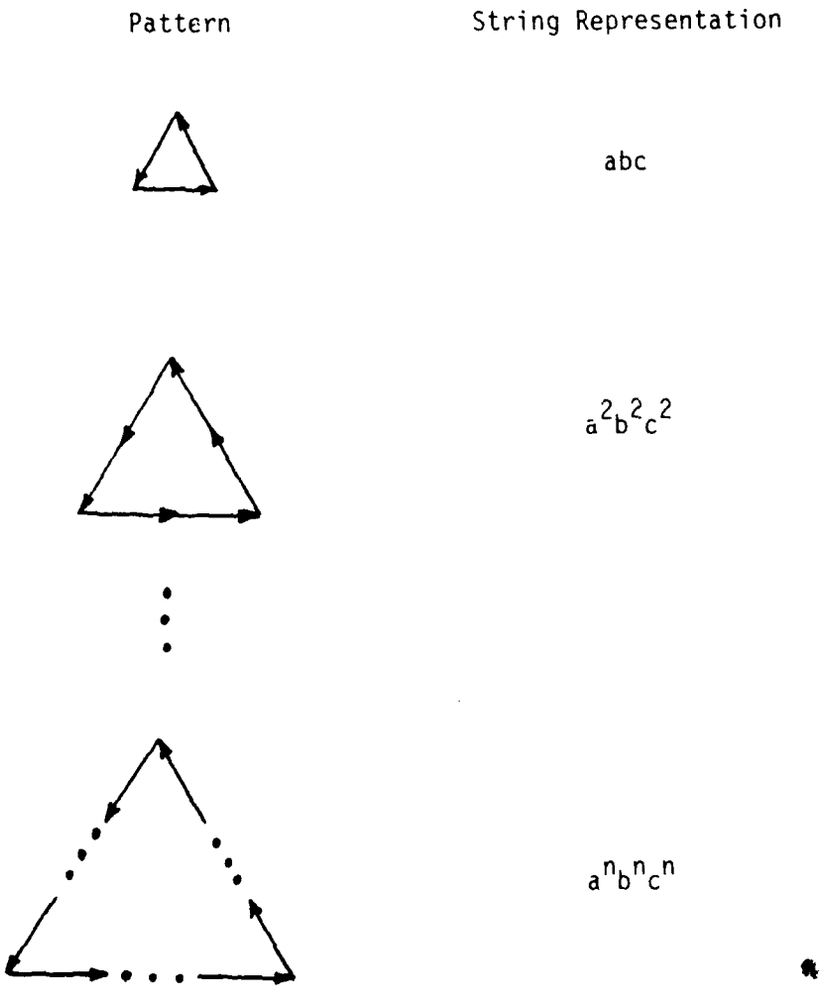


Figure 6. Patterns and their representation as strings.

form is $a^n b^n c^n$ or, in words, "at least one a followed by the same number of b's followed by the same number of c's." But the productions in the grammar G_1 do not deal in any way with values--numerical, vector, logical, or otherwise--that terminals a,b,c might take on.

The assignment of quantitative information to features in a syntactic pattern recognition formulation is accomplished via the use of attributed grammars [4-8]. The term "attributed" in this context means that we will employ a conventional syntactic grammar, but will now allow the terminals and nonterminals to have associated attributes which are assigned values by some predefined mechanism. For example, the syntactic description of certain types of 2's is given by the string α = xbvc (i.e., a convex bay followed by a concave corner). As discussed in Section 2, each feature has a given set of attributes, such as direction and length, which are quantifiers of that feature. Thus, the use of syntax establishes a given basic structure, while the attributes attach quantitative descriptions to the features forming that structure. The rules for assigning meaning to the resulting attributed structure are semantic rules analogous in form to the semantic rules described previously in connection with hierarchical feature extraction. In the case of recognition, however, the semantic rules are used for assigning meaning (i.e., valid vs. invalid character) to the overall structure. Thus, we see that the attributed-grammar approach to recognition involves three principal elements: (1) the specification of a conventional syntactic grammar, (2) the use of attributes for quantifying the features, and (3) the specification of a set of semantic rules for assigning meaning to an attributed structure.

There are two major reasons for considering attributed grammars in structural recognition. First, in most problems, it makes more sense to identify features as symbols together with their attribute values than it does to try to package all information about the primitives into a much larger set of symbols without attributes. Second, it is well known that the use of attributes and associated semantic rules can dramatically reduce the complexity of the syntactic analysis of certain classes of patterns [9].

As an illustration, we reconsider the context-sensitive language $\{y | y = a^n b^n c^n, n \geq 1\}$ generated by the grammar G_1 given earlier. The

inclusion of some very simple semantic rules allows the use of a much simpler regular grammar, as follows. The regular grammar $G_2 = (N,\Sigma,P,S)$ with nonterminal $N = \{S\}$, terminals $\Sigma = \{a,b,c\}$, productions $P = \{S^1 \rightarrow aS, S^2 \rightarrow bS, S^3 \rightarrow cS, S^4 \rightarrow c\}$ numbered for reference, and starting symbol S, generates a language $L(G_2)$ that properly contains $L(G_1)$; thus, all strings in $L(G_1)$ are syntactically correct for $L(G_2)$ as well, but there are additional strings in $L(G_2)$ that must be rejected. The semantic rules must be developed to disallow all derivation trees for strings in $L(G_2) - L(G_1)$, that is, all strings <u>not</u> of the form $a^n b^n c^n$, $n \geq 1$. In words, these rules require:

(1) all uses of production #m before production #m + 1 for $1 \leq m \leq 3$; and

(2) the same number of uses of production #1 as of production #2 as of productions #3 + #4.

This example illustrates the fact that a simple regular grammar, along with some simple semantic rules, can be made to behave as a much more powerful context-sensitive grammar. As will be seen in Section 4, this is an important point because learning algorithms for regular grammars are relatively easy to formulate.

3.2 Specification of Semantic Rules

Although, as explained in Section 4, it is possible to learn regular grammars by utilizing training samples in a grammatical inference algorithm, no automatic procedures exist for specifying semantic rules. This, however, is not a particularly serious limitation if the semantic rules are specified interactively using an approach such as the one suggested in Section 6.

The semantic rules for recognition are similar in nature to those already discussed for the hierarchical feature extractor, with the exception that they apply in general not only to single features, but also to combinations of features, as well as the production rules in a given grammar. These concepts are illustrated in the following section.

3.3 Example of Syntactic/Semantic Specification

The material discussed in the previous two sections is illustrated in this section by a syntactic/semantic specification for open-top O's.

With reference to the feature codes discussed in Section 2.10, a grammar for recognizing a large class of this type of character is given by[†]

$$G = (N,\Sigma,P,S)$$

where

$$N = \{S,A,B\}$$

$$\Sigma = \{b,c,x,v\}$$

and the productions in P are

(1) $S \rightarrow xb$	(6) $A \rightarrow vcB$
(2) $S \rightarrow xbA$	(7) $B \rightarrow xb$
(3) $S \rightarrow xcA$	(8) $B \rightarrow xc$
(4) $S \rightarrow vcS$	(9) $B \rightarrow xbS$
(5) $A \rightarrow vc$	(10) $B \rightarrow xcS$

The semantic rules for this grammar are[††]

r_1 = TRUE if the average direction of all convex bays and corners is 1 or 2 (see Fig. 3).

r_2 = TRUE if the area of each concavity is less than T_1* [average convexity area].

r_3 = TRUE if, when production (1) is applied, the degree of the bay is 3 or greater.

r_4 = TRUE if (i) production (4) is not repeated consecutively, and (ii) production (5) does not follow production (3).

r_5 = TRUE if OPENING, defined as the Euclidean distance between the terminator points, is less than T_2* [overall length].

[†]A grammar can be obtained in one of two ways: (1) heuristically by studying characters of interest, or (2) by formal grammatical inference techniques. The grammar given in this section was obtained by the former approach. Grammatical inference is discussed in Section 4.

[††]Semantic rules for recognition are denoted by lower case r's in order to differentiate them from semantic rules for the hierarchical feature extractor.

r_6 = TRUE if H/W > T_3.

where T_1 and T_2 and T_3 are constant threshold values. A given character is recognized as an open-top zero if its string representation is syntactically correct (determined by the syntactic recognizer, as discussed later) and $\cup r_i$ = TRUE, i = 1,2,...,6. In other words, the input must be both syntactically and semantically correct to be accepted.

The structure (syntax) of acceptable characters is established by the production rules. For example, the first production yields a convex bay, while the use of productions (2), (6), and (7) yields a convex bay, followed by a concave corner, followed by a convex bay.

Semantic rule r_1 establishes the acceptable direction of the overall convexity to be between 45^0 and 135^0. (The average direction of all convexities is a rugged representation of the direction of the opening.) Semantic rule r_2 precludes large concavities and thus gives a low-level guarantee of regularity. Rule r_3 establishes a minimum regularity of the boundary. For example, a bay of degree 3 closely resembles a triangle, which is deemed unacceptable for a zero. Rule r_4 similarly excludes ill-formed characters. Rule r_5 excludes large openings ("large" being measured as a fraction of overall length in order to make this rule insensitive to size.) Finally, rule r_6 precludes zeros which are short and fat beyond a given threshold. Figure 7 shows some acceptable characters and Fig. 8 shows some characters which would be rejected as being either syntactically or semantically incorrect.

3.4 Recognizer

Grammars were shown in the previous three sections to be generators of string sets. In this section, we consider the problem of recognizing a given syntactic/semantic string representation.

In terms of overall system specification--representation, learning, and recognition--the most practical approach is to employ regular grammars because of the availability of learning algorithms and the simplicity of formulation for the recognizer. As indicated in Section 3.1, the utilization of semantic rules allows considerable expansion of the pattern-representation power of regular grammars.

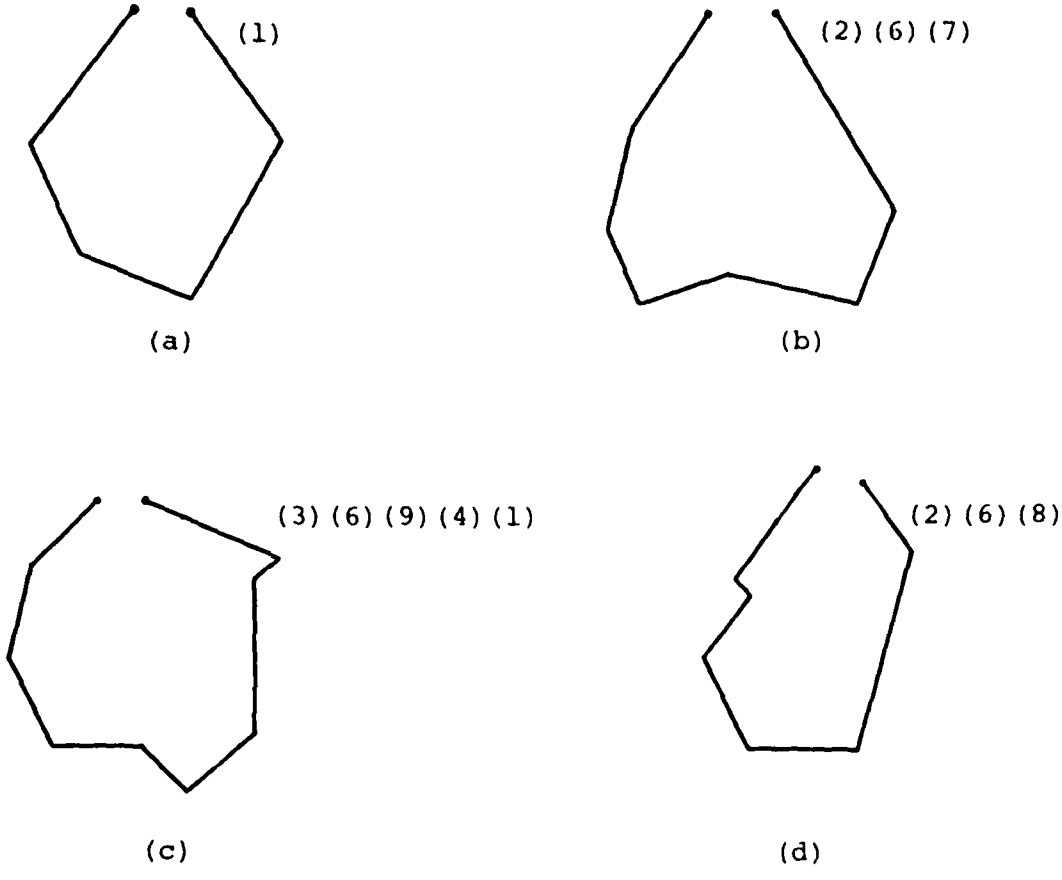


Figure 7. Characters which are both syntactically and semantically correct. The strings of numbers correspond to the sequence of productions which would generate the corresponding character.

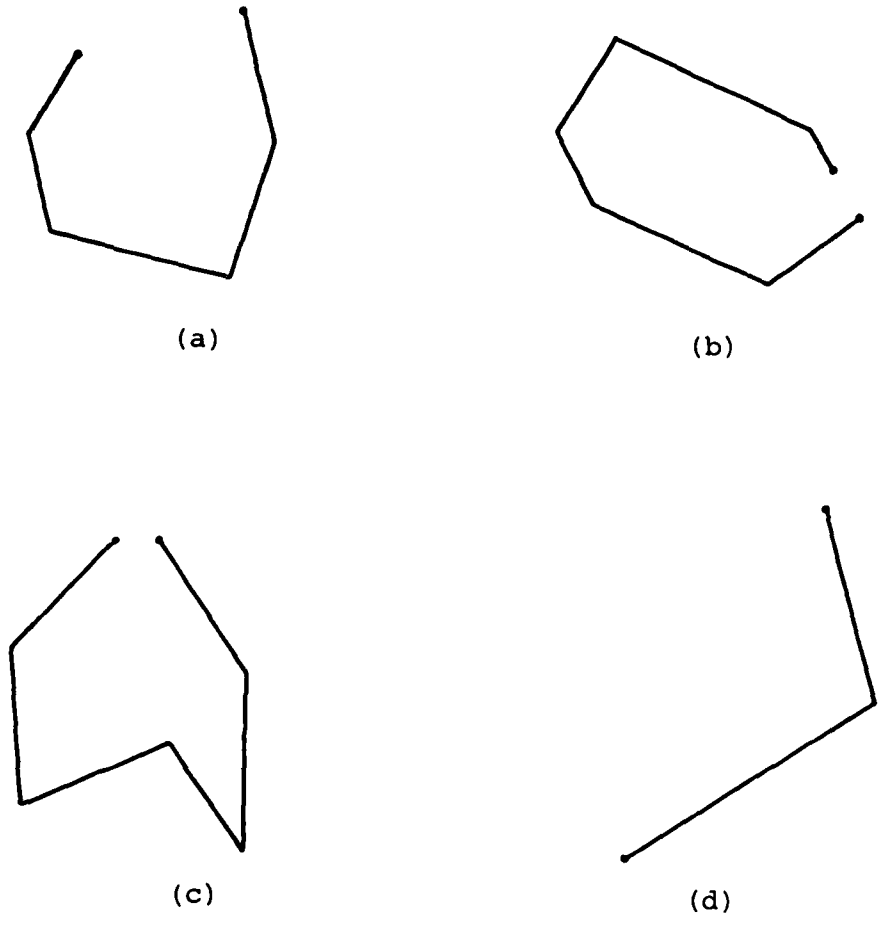


Figure 8. Examples of unacceptable characters. The patterns shown in (a), (b) and (c) are all syntactically correct, but violate semantic rules r_5, r_1, and r_2, respectively. The pattern shown in (d) is syntactically incorrect.

The formal recognizer for regular languages is the finite automaton, defined as a five-tuple $A = (Q,\Sigma,\delta,q_0,F)$, where

Q is a finite set of states,

Σ is a finite input alphabet

δ is a mapping from $Q \times \Sigma$ into 2^Q, the collection of all subsets of Q,

q_0 in Q is the starting state, and

F, a subset of Q, is a set of final or accepting states.

We say that a given string is recognized by A if, starting in state q_0, the automaton is capable of scanning the entire string and it is in one of the states of F after the last symbol in the string has been processed.

As an illustration of this notation, consider the automaton $A = (Q,\Sigma, \delta,q_0,F)$ with

$$Q = \{q_0,q_1\},\ \Sigma = \{a,b\},\quad F = \{q_1\}$$

and mappings

$$\delta(q_0,a) = \{q_0\}$$
$$\delta(q_0,b) = \{q_1\}$$
$$\delta(q_1,a) = \delta(q_1,b) = \phi$$

where ϕ is the null set (undefined states in this case).

A finite automaton is conveniently represented by its state transition diagram, a directed graph whose nodes, corresponding to states, are connected by arcs that are labeled with input symbols which cause transitions. By convention, all final states are denoted by double circles and the starting state is designated by an entering arrow.

The state transition diagram for the example just given is shown in Fig. 9. It is noted that this automaton remains in state q_0 for any number of input a's, and makes a transition to the final state when a symbol b occurs in the string. Thus, the language accepted by the automaton consists of the set of strings of the form $\{a^nb\}$, $n \geq 0$. All other strings are rejected. For example, input abb is not accepted because $\delta(q_0,a) = \{q_0\}$, $\delta(q_0,b) = \{q_1\}$, but $\delta(q_1,b) = \phi$, causing A to halt because it is unable to complete processing the entire string.

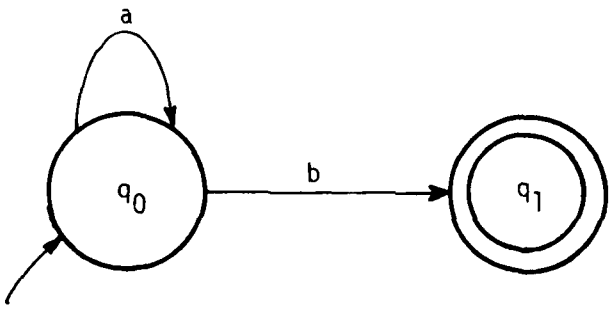


Figure 9. State transition diagram.

In order to incorporate the semantic specifications discussed in the previous section into the recognition procedure, it is important to establish the relationship between a grammar and its corresponding automaton. This relationship is based on a fundamental theorem from formal language theory which states that a language is recognized by a finite automaton iff it is generated by a regular grammar [3].

Given a regular grammar $G = (N,\Sigma,P,X_0)$, where X_0 is the starting symbol, the corresponding finite automaton $A = (Q,\Sigma,\delta,q_0,F)$ is specified as follows.† Suppose the nonterminal set N is composed of the starting symbol X_0 and n additional nonterminals $X_1,X_2,\ldots,X_n$. Then, the state set Q of A is formed by n + 2 states $\{q_0,q_1,\ldots,q_n,q_{n+1}\}$ such that q_i corresponds to X_i for $0 \leq i \leq n$, and q_{n+1} is an additional state such that $F = \{q_{n+1}\}$. The set of input symbols of A is the same as Σ in G, and the δ mapping is defined by two rules based on the productions of G, as follows: For $0 \leq i \leq n$, $0 \leq j \leq n$, and any a in Σ,

1) if $X_i \rightarrow aX_j$ is in P, then $\delta(q_i,a)$ contains q_j, and
2) if $X_i \rightarrow a$ is in P, then $\delta(q_i,a)$ contains q_{n+1}.

As an illustration of this procedure, consider the grammar for open-top 0's given in Section 3.3. This grammar is easily converted to regular form by defining the following symbols

$$b_1 = xb$$
$$b_2 = xc$$
$$b_3 = vc$$

Using these symbols and the above notation for the nonterminals, (i.e., $S = X_0$, $A = X_1$, and $B = X_2$), the grammar becomes

$$G = (N,\Sigma,P,X_0)$$

where

$$N = \{X_0,X_1,X_2\}$$
$$\Sigma = \{b_1,b_2,b_3\}$$

---

†A similar procedure exists for obtaining the regular grammar corresponding to a given finite automaton [3].

and the productions in P are

(1) $X_0 \rightarrow b_1$	(6) $X_1 \rightarrow b_3X_2$
(2) $X_0 \rightarrow b_1X_1$	(7) $X_2 \rightarrow b_1$
(3) $X_0 \rightarrow b_2X_1$	(8) $X_2 \rightarrow b_2$
(4) $X_0 \rightarrow b_3X_0$	(9) $X_2 \rightarrow b_1X_0$
(5) $X_1 \rightarrow b_3$	(10) $X_2 \rightarrow b_2X_0$

Based on the preceding discussion, the automaton $A = (Q,\Sigma,\delta,q_0,F)$ corresponding to this grammar has the state set

$$Q = \{q_0,q_1,q_2,q_3\}$$

symbol set

$$\Sigma = \{b_1,b_2,b_3\}$$

and final state set

$$F = \{q_3\}$$

The mappings are obtained by applying the two rules given above to the productions of G. For example, production (5) is of the form shown in rule 2, so that $\delta(q_1,b_3) = \{q_3\}$, while production (6) is of the form shown in rule 1, which gives $\delta(q_1,b_3) = \{q_2\}$. Thus, the combined mapping is $\delta(q_1,b_3) = \{q_2,q_3\}$.[†] Following this procedure yields the following mappings corresponding to the ten productions of G:

$$\delta(q_0,b_1) = \{q_1,q_3\}$$
$$\delta(q_0,b_2) = \{q_1\}$$
$$\delta(q_0,b_3) = \{q_0\}$$
$$\delta(q_1,b_3) = \{q_2,q_3\}$$
$$\delta(q_2,b_1) = \{q_0,q_3\}$$
$$\delta(q_2,b_2) = \{q_0,q_3\}$$

[†]The fact that there are two possible transitions for this state with the same input symbol indicates that this is a non-deterministic automaton. As shown in [3], such an automaton is easily made deterministic by introducing additional states.

All other transitions, for example $\delta(q_1,b_0)$, yield the null set, indicating an unacceptable input string. The state transition diagram is shown in Fig. 10.

The automaton just obtained is a recognizer based on the structure (syntax) of strings corresponding to open-top O's. In order to incorporate the semantic rules discussed in Section 3.3, we first determine if a given string is syntactically correct (i.e., if it is accepted by the automaton). If it is, the semantic rules are tested using the procedure discussed in Section 3.3. The only exception is that, instead of productions, we use the corresponding mappings in the automaton. For example, semantic rule r_4 would now read: r_4 = TRUE if mapping $\delta(q_0,b_3)$ is not repeated consecutively, and (ii) mapping $\delta(q_1,b_3) = \{q_3\}^{\dagger}$ does not follow mapping $\delta(q_0,b_2)$. This type of information can be easily incorporated into the recognition process in the form of a history of automaton transitions as a string is processed.

---

†Note that since the automaton is non-deterministic the transition actually corresponding to production (5) must be used in testing this semantic rule.

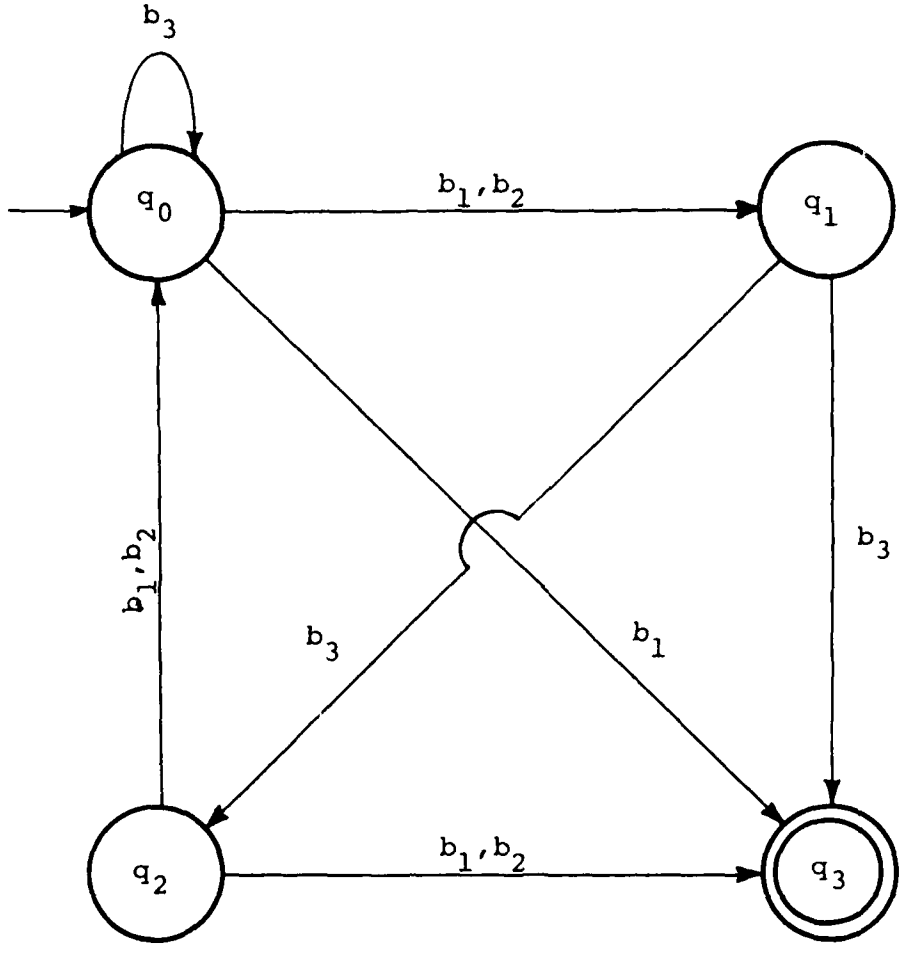


Figure 10. Recognizer (finite-automaton) for open-top zeros.

IV. LEARNING

4.1 Background

As indicated in the previous section, one of the principal advantages of using syntactic/semantic formulations in pattern recognition is that it often allows utilization of grammars which are considerably simpler than those that would be required if only syntax were employed. In particular, the use of regular grammars enjoys the distinct advantage of having the best-developed learning algorithms. The algorithm presented in this section relies only on one input parameter, and its behavior as a function of this parameter is fully understood. As will be shown in the following discussion, this procedure learns the structure of a finite automaton directly from a sample set of training patterns expressed in the form of strings.

4.2 Learning Algorithm

Let R be a set of pattern strings (including the empty string) and let z be a string in Σ^* such that zw is in R for some w in Σ^*.[†] Given a non-negative integer k, the k-tail of z with respect to R is defined as the set $\{w|zw \text{ in } R, |w| \leq k\}$. In other words, the k-tail of a string z is a set consisting of all the strings w subject to the conditions that, for any particular w, the string zw is in R and the length of w is less than or equal to k. For notational convenience, we denote the k-tail set as

$$h(z,R,k) = \{w|zw \text{ in } R, |w| \leq k\}$$

This notation clearly shows the functional dependence of the k-tail on z, R, and k.

Using the k-tail definition, an automaton corresponding to R and a given value of k is obtained by means of the following procedure [3]:

(1) Σ is formed from all the different symbols in R.

[†] Σ^* is the notation used to represent the set of all strings formed from symbols of Σ, including the empty string.

(2) The initial state is given by

$$q_0 = h(\lambda,R,k)$$

where λ is the empty string.

(3) The set of states is given by

$$Q = \{q \mid q = h(z,R,k) \text{ for } z \text{ in } \Sigma^*\}.$$

(4) the final state set is given by

$$F = \{q \mid q \text{ in } Q, \ \lambda \text{ in } q\}.$$

(5) The mappings from a state q with an input symbol "a" are given by

$$\delta(q,a) = \{q' \mid q' \text{ in } Q,\ q' = h(za,R,k),\ q = h(z,R,k)\}$$

This procedure for obtaining an automaton is best explained by an example. Suppose that R = {a,ab,abb} and we let k = 1. From Step 1, we have Σ = {a,b}. Step 2 gives the starting state as $q_0 = h(\lambda,R,1)$. From the above definition of h,

$$\begin{aligned} h(\lambda,R,1) &= \{w \mid \lambda w \text{ i } R,\ |w| \leq 1\} \\ &= \{a\} \\ &= q_0 \end{aligned}$$

In other words, since $|\lambda| = 0$, the only string λw that is in R, and has length less than or equal to 1, is a. Thus, the set {a} is defined as corresponding to state q_0.

The set Q of states is obtained from Step 3 by changing z. As shown above, when $z = \lambda$ we have

$$\begin{aligned} h(\lambda,R,1) &= \{a\} \\ &= q_0 \end{aligned}$$

Next, we select z = a and obtain

$$h(a,R,1) = \{w \mid aw \text{ in } R,\ |w| \leq 1\}$$
$$= \{\lambda, b\}$$
$$= q_1$$

In this case the only strings w that can be appended to z = a with zw being in R and $|w| \leq 1$ are λ and b. We define this new set as state q_1.

Next we consider the string z = ab and obtain

$$h(ab,R,1) = \{w \mid abw \text{ in } R,\ |w| \leq 1\}$$
$$= \{\lambda, b\}$$
$$= q_1$$

which does not produce a new state. The next string is z = abb. This yields,

$$h(abb,R,1) = \{\lambda\}$$
$$= q_2$$

Other strings z in Σ^* will yield, in this case, strings zw that are not in R, giving rise to a fourth state, denoted by q_ϕ, which corresponds to the condition that h is the null set. Therefore, we have the state set

$$Q = \{q_0, q_1, q_2, q_\phi\}$$

According to Step 4, the final state set is given by

$$F = \{q \mid q \text{ in } Q, \lambda \text{ in } q\}$$
$$= \{q_1, q_2\}$$

Note that, since both q_1 and q_2 contain λ, these two states are in the final state set.

Finally, the mappings are obtained using Step 5. Starting with q_0, we obtain

$$\delta(q_0,a) = \{q' \mid q' \text{ in } Q,\ q' = h(\lambda a,R,1),\ q_0 = h(\lambda,R,1)\}$$

Since $q' = h(\lambda a,R,1) = h(a,R,1) = q_1$, we have

$$\delta(q_0,a) = q_1$$

Similarly,

$$\begin{aligned}\delta(q_0,b) &= \{q' \mid q' \text{ in } Q,\ q' = h(\lambda b,R,1) \\ &\qquad\qquad q_0 = h(\lambda,R,1)\} \\ &= q_\phi\end{aligned}$$

The second step follows from the fact that $q' = h(b,R,1) = q_\phi$.

Next we consider transitions from q_1, which has two representations, $h(a,R,1)$ and $h(ab,R,1)$. Using the first representation yields

$$\begin{aligned}\delta(q_1,a) &= \{q' \mid q' \text{ in } Q,\ q' = h(aa,R,1), \\ &\qquad\qquad q_1 = h(a,R,1)\} \\ &= q_\phi\end{aligned}$$

Using the second representation we obtain

$$\begin{aligned}\delta(q_1,a) &= \{q' \mid q' \text{ in } Q,\ q' = h(aba,R,1), \\ &\qquad\qquad q_1 = h(ab,R,1)\} \\ &= q_\phi\end{aligned}$$

The transitions from q_1 with an input of b are similarly given by

$$\begin{aligned}\delta(q_1,b) &= \{q' \mid q' \text{ in } Q,\ q' = h(ab,R,1), \\ &\qquad\qquad q_1 = h(a,R,1)\} \\ &= q_1\end{aligned}$$

and

$$\begin{aligned}\delta(q_1,b) &= \{q' \mid q' \text{ in } Q,\ q' = h(abb,R,1), \\ &\qquad\qquad q_1 = h(ab,R,1)\} \\ &= q_2\end{aligned}$$

Following this procedure for q_2 and q_ϕ yields the following mappings:

$$\delta(q_2,a) = q_\phi$$
$$\delta(q_2,b) = q_\phi$$
$$\delta(q_\phi,a) = q_\phi$$
$$\delta(q_\phi,b) = q_\phi$$

The automaton just obtained from the strings in R is shown in Fig. 11. It is of interest to note that the automaton recognizes strings of the form ab^n, $n \geq 0$. This is a reasonable generalization of the structure present in the strings of the learning set: R = {a,ab,abb}.

4.3 Properties of the Inferred Automaton

Given a specific string set for learning, the procedure discussed in the previous section has a very important characteristic: it depends only on the parameter k. Furthermore, the behavior of the learning method is known to have some useful properties as a function of k. Letting L[A(R,k)] represent the language accepted by the inferred automaton, A, for a specific R and k, these properties may be stated as follows [3]:

(1) For any $k \geq 0$, R is a subset of L[A(R,k)].

(2) If $k \geq m$, the length of the longest string in R, then L[A(R,m)] = R.

(3) If k = 0, then $L[A(R,0)] = \Sigma^*$.

(4) L[A(R,k+1)] is a subset of L[A(R,k)].

The first property guarantees that, as a minimum, A will accept all the strings in R. Property 2 guarantees that A will accept only R if we set $k \geq m$. Property 3 states that k = 0 gives a useless result: an automaton that accepts all strings composed of symbols from Σ. Finally, Property 4 simply states that increasing k increases the selectivity of the recognizer. From these properties, it is easily seen that k must be in the range $0 < k \leq m$.

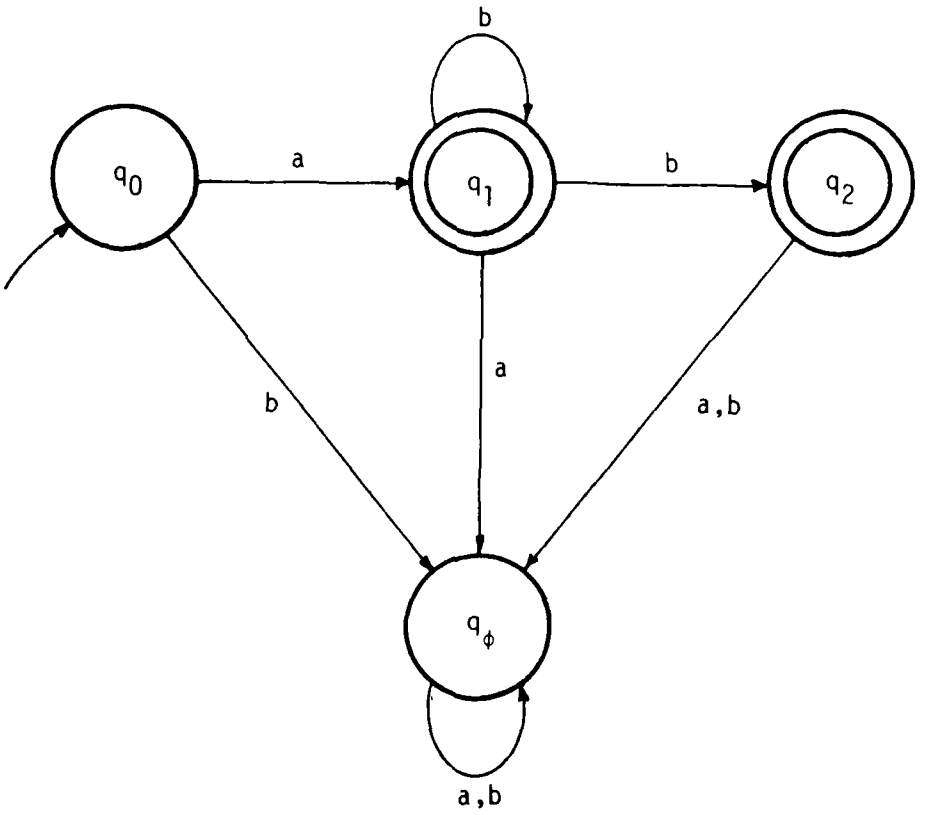


Figure 11. Inferred automaton.

V. CHECKING FOR CLASS SEPARABILITY

One of the most important issues in the design of pattern recognition systems is to have an a priori idea of how well a set of selected features discriminate between different pattern classes. In the present problem, this is equivalent to establishing whether or not two or more automata (of different classes) recognize the same subset of strings. In other words, a string that is recognized by more than one automaton is a result of overlapping pattern classes. As discussed below, another important property associated with using finite-state automata as recognizers is that checking for this type of overlap is reasonably straightforward.

Suppose we have two regular languages L_1 and L_2, recognized respectively by deterministic finite automata $A_1 = (Q_1, \Sigma_1, \delta_1, \hat{q}_0, F_1)$ and $A_2 = (Q_2, \Sigma_2, \delta_2, \hat{\hat{q}}_0, F_2)$; that is, $L_1 = L(A_1)$ and $L_2 = L(A_2)$. Then an automaton A such that $L(A) = L(A_1) \cap L(A_2)$ is given by

$$A = (Q, \Sigma, \delta, q, F)$$

where

(1) $Q = Q_1 \times Q_2 = \{(\hat{q}, \hat{\hat{q}}) \mid \hat{q} \in Q_1, \hat{\hat{q}} \in Q_2\}$

(2) $\Sigma = \Sigma_1 \cup \Sigma_2$

(3) δ is a mapping from $Q \times \Sigma$ onto Q such that, for any symbol "a" from Σ,

$$\delta((\hat{q}, \hat{\hat{q}}), a) = (\delta_1(\hat{q}, a), \delta_2(\hat{\hat{q}}, a))$$

(4) $q_0 = (\hat{q}_0, \hat{\hat{q}}_0)$ is the starting state

(5) $F = \{(\hat{q}, \hat{\hat{q}}) \mid \hat{q} \in F_1, \hat{\hat{q}} \in F_2\}$

This automaton accepts an input string if and only if both A_1 and A_2 accept it. In other words, A recognizes all string in the intersection of L_1 and L_2. The notation introduced above is clarified by the following example.

Consider the automata

$$A_1 = (Q_1, \Sigma_1, \delta_1, \hat{q}_0, F_1)$$

with

$$Q_1 = \{\hat{q}_0, q_1, q_2\}$$

$$\Sigma_1 = \{a,b\}$$

δ_1:

$$\delta_1(\hat{q}_0,a) = \hat{q}_0 \qquad \delta_1(\hat{q}_0,b) = q_1$$
$$\delta_1(q,a) = q_2 \qquad \delta_1(q_1,b) = q_1$$
$$\delta_1(q_2,a) = q_2 \qquad \delta_1(q_2,b) = q_2$$

$$F_1 = \{q_1\}$$

and

$$A_2 = (Q_2, \Sigma_2, \delta_2, \hat{\hat{q}}_0, F_2)$$

with

$$Q_2 = \{\hat{\hat{q}}_0, q_4, q_5\}$$

$$\Sigma_2 = \{a,b\}$$

δ_2:

$$\delta_2(\hat{\hat{q}}_0,a) = \hat{\hat{q}}_0 \qquad \delta_2(\hat{\hat{q}}_0,b) = q_4$$
$$\delta_2(q_4,a) = q_5 \qquad \delta_2(q_4,b) = q_5$$
$$\delta_2(q_5,a) = q_5 \qquad \delta_2(q_5,b) = q_5$$

$$F_2 = \{q_5\}$$

The state transition diagrams for A_1 and A_2 are shown in Figs. 12(a) and (b). The languages recognized by these automata are, respectively,

$$L_1 = L(A_1) = a^n\, b\, b^m \qquad n,m \geq 0$$
$$L_2 = L(A_2) = a^n\, b(a + b)(a + b)^m \qquad n,m \geq 0$$

where the "+" indicates "or". That is, an element of $L(A_1)$ is a (possibly empty) string of a's of length n ($n \geq 0$), followed by a string of at least one b, and an element of $L(A_2)$ is a (possibly empty) string of a's followed by at least one b, followed by a string of at least one a or b, and any combination of a's or b's thereafter.

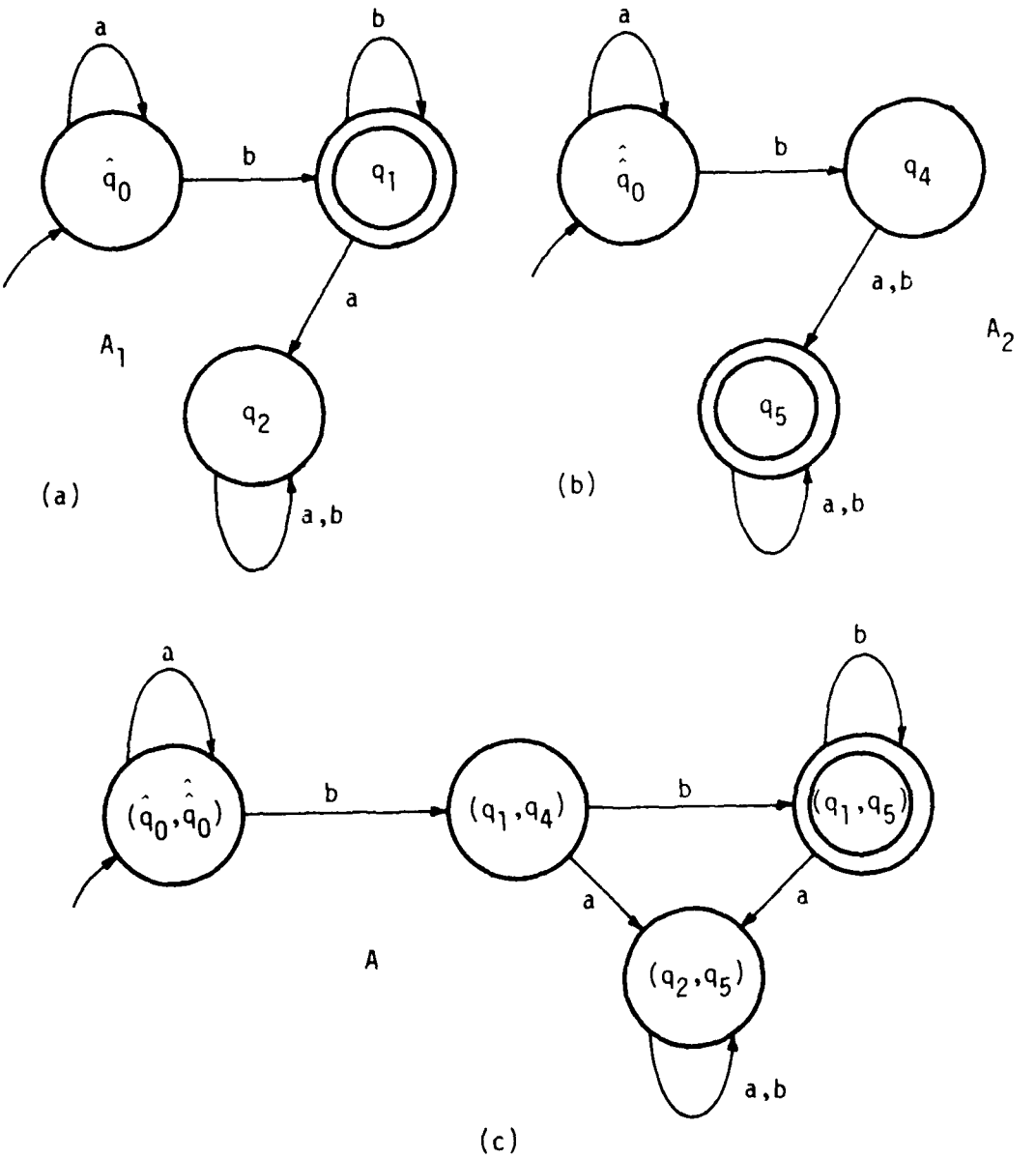


Figure 12. State transition diagrams for A_1, A_2, and A.

In order to form the automaton A which recognizes $L(A_1) \cap L(A_2)$, we proceed as follows. Using the notation introduced above,

$$A = (Q, \Sigma, \delta, q_0, F)$$

where the starting state is $q_0 = (\hat{q}_0, \hat{\hat{q}}_0)$, and

$$Q = Q_1 \times Q_2 = \{(\hat{q}_0, \hat{\hat{q}}_0), (\hat{q}_0, q_4), (\hat{q}_0, q_5), (q_1, \hat{\hat{q}}_0), (q_1, q_4), (q_1, q_5), (q_2, \hat{\hat{q}}_0), (q_0, q_4), (q_0, q_5)\}$$

$$\Sigma = \Sigma_1 \cup \Sigma_2 = \{a, b\}$$

δ:

$$\delta((\hat{q}_0, \hat{\hat{q}}_0), a) = (\hat{q}_0, \hat{\hat{q}}_0) \qquad \delta((\hat{q}_0, \hat{\hat{q}}_0), b) = (q_1, q_4)$$

$$\delta((q_1, q_4), a) = (q_2, q_5) \qquad \delta((q_1, q_4), b) = (q_1, q_5)$$

$$\delta((q_2, q_5), a) = (q_2, q_5) \qquad \delta((q_2, q_5), b) = (q_2, q_5)$$

$$\delta((q_1, q_5), a) = (q_2, q_5) \qquad \delta((q_1, q_5), b) = (q_1, q_5)$$

$$F = \{(q_1, q_5)\}$$

It is noted that the set Q consists of all ordered pairs of states in Q_1 and Q_2. Also the notation (q_i, q_j) refers to a single state of A, and is used to accentuate the fact that a state of A arises from states q_i and q_j in A_1 and A_2, respectively. Thus, A may be viewed as implementing A_1 and A_2 simultaneously in parallel. Automaton A acts as A_1 and A_2 driven by the same input, with A accepting an input if and only if both A_1 and A_2 accept it.

With reference to the above δ mappings, it is noted that no transition functions were specified for $(\hat{q}_0, q_4), (q_1, \hat{\hat{q}}_0), (\hat{q}_0, q_5), (q_1, \hat{q}_0)$, and (q_2, q_4). The reason for this is easily explained by noting the fact (see Fig. 12) that these five composite states are combinations of states in A_1 and A_2 at least one of which is unreachable starting from $\hat{q}_0$ and $\hat{\hat{q}}_0$, respectively. This implies that no input string exists which can cause A to reach any of the five states listed above by starting at its initial state $(\hat{q}_0, \hat{\hat{q}}_0)$.

These unreachable states can be eliminated from the state set Q without affecting L(A).

The language

$$L(A) = L(A_1) \cap L(A_2) = a^n\ b\ b\ b^m \qquad n,m \geq 0$$

consists of strings which have the form of a series (possibly empty) of a's followed by a string of at least two b's, which are the strings which are accepted by both A_1 and A_2.

The procedure just discussed has two important implications. First, if $L(A_1)$ and $L(A_2)$ are disjoint, then L(A) will be the empty set, indicating perfect recognition by A_1 and A_2. Second, if L(A) is not empty, the structure of the strings in the overlapping subset can easily be examined. As indicated in the following section, one approach for eliminating the overlap is to use semantics. It is also noted that the method is easily extended to a multiclass situation simply by considering two classes at a time.

VI. ORGANIZATION OF THE SYNTACTIC/SEMANTIC CHARACTER RECOGNITION SYSTEM

The purpose of this section is to organize the material presented in the previous sections in the form of a system which utilizes both syntax and semantics in the recognition process.

The basic approach proposed for designing (training) the recognition system is shown diagramatically in Fig. 13, where the dashed lines indicate interactive user inputs. The various stages of this approach are discussed in the following paragraphs.

The training set consists of a set of thinned characters of known classification. The hierarchical feature extraction and attribute assignment stage is based on the methods discussed in Section 2. It is noted that, although the features to be extracted are well defined, the semantic rules of some of these features require the specification of one or more thresholds. For example, the semantic rules for lake features require the specification of thresholds $T_1(4)$ and $T_2(4)$. The procedure for doing this is to compute this particular feature for all appropriate characters in the training set and then to select the thresholds as the minimum values which encompass all acceptable lakes, with the degree of acceptability being established by the user via a display examination of these features. In other words, the user must determine what constitutes an acceptable lake feature and the thresholds are used to place limits on the class of acceptable features. The selection of thresholds for other features is carried out in a similar manner.

The next step in the training procedure is to arrange the extracted features in the form of a string, as discussed in Section 2.10. These strings are then fed into the grammatical inference stage, where an automaton is generated for each class using the method discussed in Section 4. It is noted that the only parameter required by the inference algorithm is a value of k to establish the k-tail.

Given a value of k, the resulting automata are then used to recognize the training set. It is expected that this stage of the process will require the most intensive user interaction. The automata generated by

the inference algorithm provide the basic syntax (structure) of the training set subject to the limitations inherent in regular grammars. It is unlikely that this formalism by itself will be sufficient to completely classify the training set correctly. The two tools available to increase discrimination are the check for class separability shown as the next stage in Fig. 13, and the use of semantic rules. The set of overlapping strings in any two classes can easily be established using the procedure discussed in Section 5. This information can be used to study the structure of the strings that are not uniquely recognizable, and semantic rules can be introduced to resolve the conflicts, as discussed in Section 3. It is noted that the use of a display to show the appropriate automata and to highlight the state transitions followed in recognizing a given string will be a valuable aid in establishing the necessary semantic information.

The design of the syntactic/semantic recognizer is complete once the training set is recognized with acceptable accuracy. During automatic operation, the structure of the system consists of the stages shown in Fig. 14. In this mode of operation a character can be rejected prior to going into the recognition stage if its features and attributes are outside the learned thresholds or fail to satisfy the corresponding semantic rules. If a character passes this test, it is fed into the recognizer (automata). At this point it is assigned to a character class or rejected if it fails to be accepted by a unique automaton based on the syntactic/semantic information developed for this stage during the training phase.

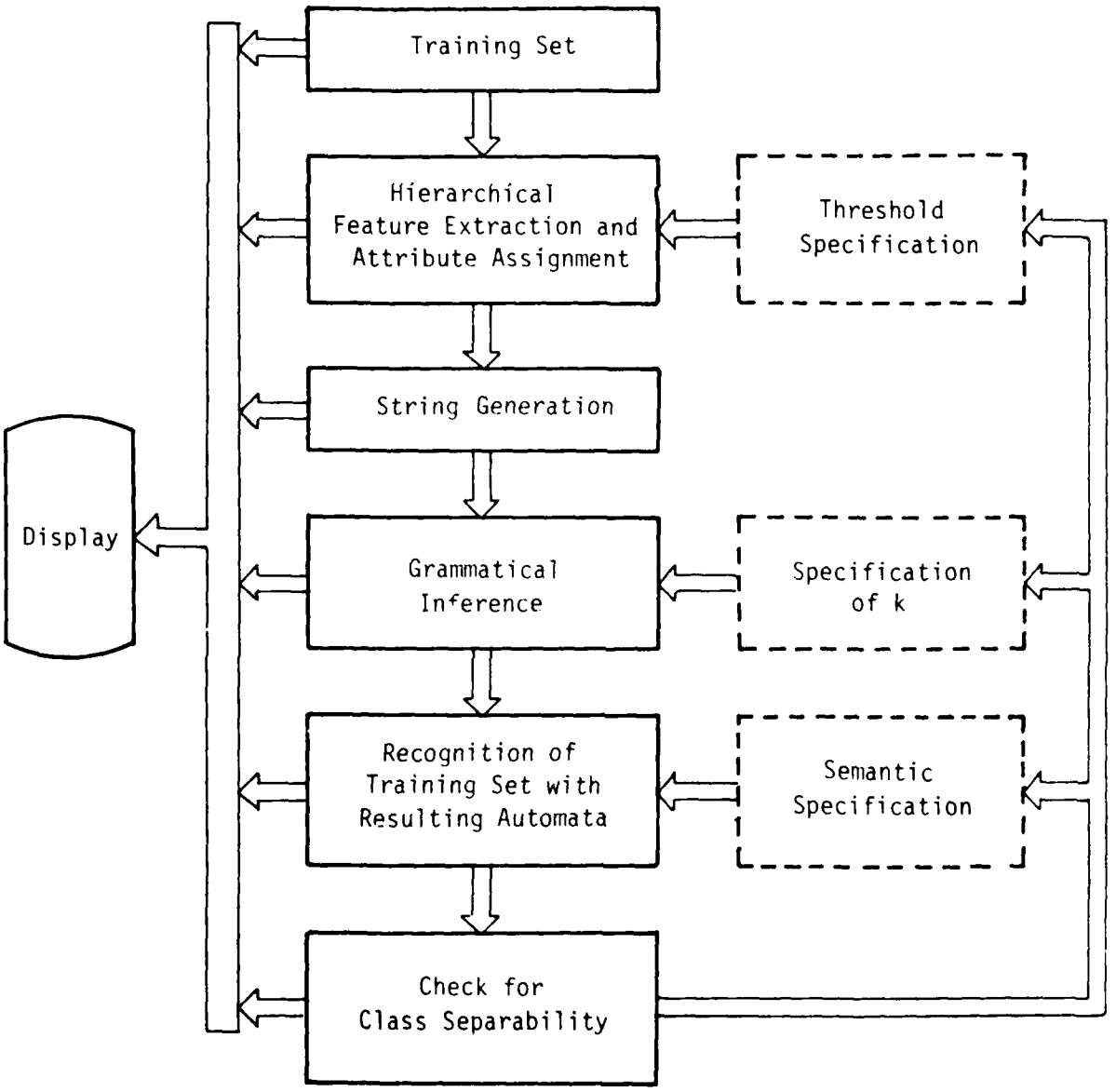


Figure 13. Structure of the syntactic/semantic recognizer during training. Dashed lines indicate interactive inputs.

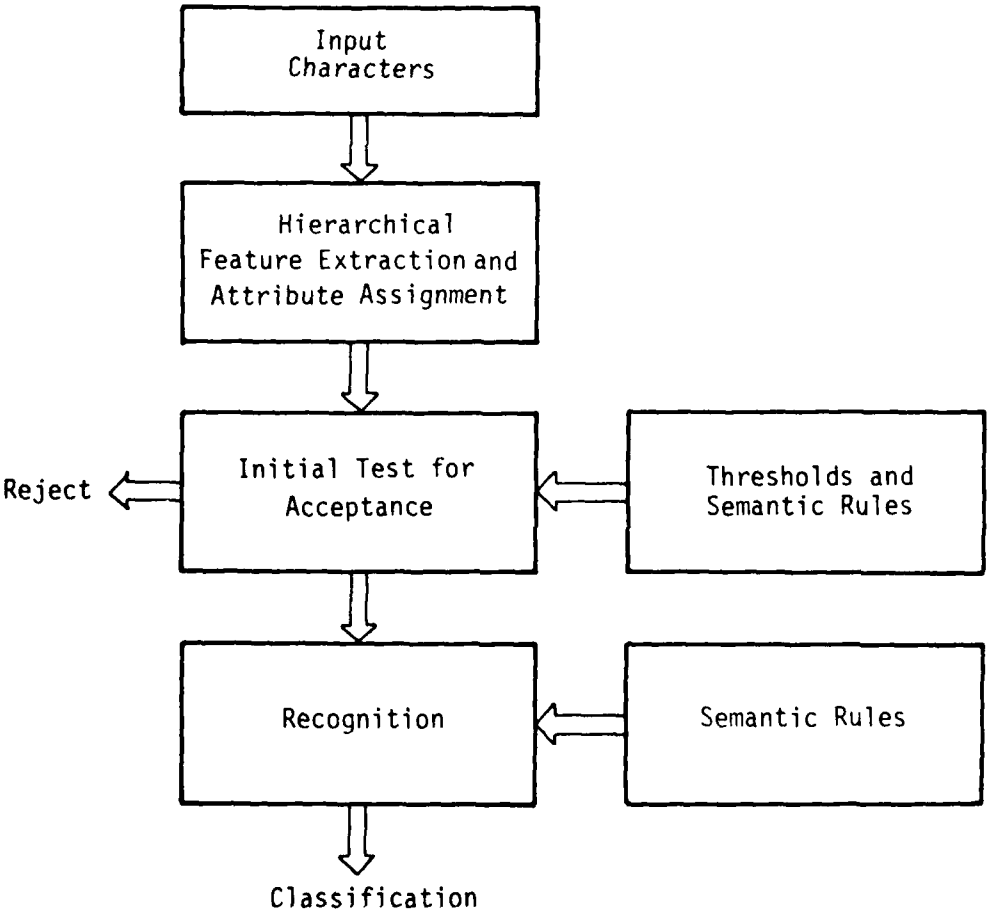


Figure 14. Syntactic/semantic recognizer in automatic operation.

VII. CONCLUSIONS AND RECOMMENDATIONS

The material discussed in the previous sections represents a unified approach for the development of a syntactic/semantic character recognition system. The most important aspects of this approach are: (1) a hierarchical, semantics-based feature extractor, (2) a formulation that leads to representations that can be handled with string grammars, (3) the use of semantics in the recognition process, (4) the use of a procedure for studying class separability, and (5) a proposed interactive approach which combines automatic syntactic processing with user-defined semantic rules.

Although the overall system structure has been developed in some detail, there are a number of areas that require further investigation. In particular, we recommend that the following tasks be carried out as the next step in this project.

Task 1. Extension of semantic rules for the hierarchical feature extractor. The semantic rules proposed in the report are preliminary. This task will consist of extending and refining the semantic rules for feature description in the context of the NORDA OCR system.

Task 2. Evaluate the parsing approach to recognition. This task will investigate the formulation of a parsing (vs. automaton) approach to recognition. Parsing algorithms are generally faster and this task will address the problem of incorporating semantic rules into the parsing process.

Task 3. Extend the semantic rules for the syntactic/semantic recognizer. Considerable work remains to be done in proposing semantic rules for the recognition stage. This task will address the problem of specifying a set of semantic rules for each of the ten numeral classes, with possible extension to alphanumerics.

Task 4. Extend the syntactic/semantic approach to border-oriented features. Work done to date on the syntactic/semantic approach has been focused on skeleton-oriented features. It is known that border-oriented features can be very useful in situations

involving characters such as filled-in 8's. This task will address the extension of the hierarchical feature extractor and the syntactic/semantic recognizer for handling border features.

Task 5. Refine the interactive approach used in the design of the recognition system. The approach used in the specification of semantic rules is highly interactive. This task will consist of developing a specific formulation for the implementation of this approach, including techniques for user specification of relevant parameters.

REFERENCES

1. Gonzalez, R. C., "Evaluation of the CHITRA Character Recognition System and Development of Feature Extraction Algorithms," Report on NORDA Contract N00014-80-M-0030, Digital Decision Systems, Inc., May, 1980.

2. Tou, J. T. and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, Reading, MA, 1974.

3. Gonzalez, R. C. and M. G. Thomason, *Syntactic Pattern Recognition: An Introduction*, Addison-Wesley, Reading, MA, 1978.

4. Thomason, M. G., "Syntactic Methods in Pattern Recognition," in *Pattern Recogniton Theory and Applications*, (J. Kittler, K. S. Fu, and L. F. Pau, eds.), D. Reidel Publishing Co., Dordrecht, Holland, pp. 119-138, 1982.

5. Davis, C. S. and T. C. Henderson, "Hierarchical Constraint Processes for Shape Analysis," *IEEE Trans. PAMI*, vol. PAMI-3, pp. 265-277, 1981.

6. Tang, G. Y. and T. S. Huang, "A Syntactic-Semantic Approach to Image Understanding and Creation," *IEEE Trans. PAMI*, vol. PAMI-1, pp. 135-144, 1979.

7. Tang, G. Y. and T. S. Huang, "Using the Creation Machine to Locate Airplanes on Aerial Photos," *Pat. Recog.*, vol. 12, 431-442, 1980.

8. You, K. C. and K. S. Fu, "A Syntactic Approach to Shape Recognition Using Attributed Grammars," *IEEE Trans. SMC*, vol. SMC-9, pp. 334-335, 1979.

9. Pyster, A. and H. W. Buttlemann, "Semantic Syntax-Directed Translation," *Info. and Control*, vol. 36, pp. 320-361, 1978.

10. Fu, K. S. and T. L. Booth, "Grammatical Inference: Introduction and Survey," *IEEE Trans. SMC*, vol. SMC-5, pp. 95-111 and 409-423, 1975.

11. Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Pub. Co., Palo Alto, CA, 1980.

12. Winston, P. H., *Artificial Intelligence*, Addison-Wesley, Reading, MA, 1977.

13. Simon, H. A., "Studying Human Intelligence by Creating Artificial Intelligence," *Amer. Sci.*, vol. 69, pp. 300-309, 1981.

14. Aho, A. V. and J. D. Ullman, *The Theory of Parsing, Translation, and Compiling*, vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1972.

APPENDIX A

The following discussion deals with elliptical symmetry. This type of feature is useful in character recognition for determining the quality of characters composed, either partially or entirely, of elliptical segments, such as 9's and 0's. Two procedures are developed below. The first is based on a minimum-error elliptical fit, while the second applies to any type of symmetry about two principal axes. Both methods are independent of rotation.

A.1 Procedure 1

Given K sets of points, E_i, i=1,2,...,K, with set E_i containing $\#E_i$ points, the following procedure individually measures elliptical symmetry about two principal orthogonal axes for each set. In terms of character recognition, each set E_i contains the coordinate points of, for example, the skeleton of a character, and i ranges over the number of characters (K) to be processed.

(a) Let $\{x_n, y_n\}$, n = 1,2,...,$\#E_i$, represent the coordinates of all points in E_i.

(b) Define the column vectors $\underline{z}_n = (x_n, y_n)'$, where the prime (') indicates transposition.

(c) Compute the 2 x 2 covariance matrix

$$\underline{C}_i = \frac{1}{\#E_i} \sum_{n=1}^{\#E_i} \underline{z}_n \underline{z}_n' - \underline{m}_i \underline{m}_i'$$

where

$$\underline{m}_i = \frac{1}{\#E_i} \sum_{n=1}^{\#E_i} \underline{z}_n$$

(d) Compute the two orthogonal eigenvectors and corresponding eigenvalues of $\underline{C}_i$. (Since the matrix is real and symmetric, the existence of orthogonal eigenvectors is guaranteed. Almost any scientific package will contain a subroutine for

computing the eigenvectors and eigenvalues of a real, symmetric matrix). These two eigenvectors, denoted by $\underline{e}_1 = (a,b)'$ and $\underline{e}_2 = (c,d)'$, point in the directions of principal data spread, subject to the orthogonality constraint. The amount of spread is proportional to the eigenvalues and is assumed for notational convenience that the largest eigenvalue corresponds to $\underline{e}_1$.

(e) The equation of an ellipse in the (x,y) plane centered about $\underline{m}_i$ and with $\underline{e}_1$ and $\underline{e}_2$ as the principal axes is given by

$$(\underline{z} - \underline{m}_i)' \underline{C}_i^{-1} (\underline{z} - \underline{m}_i) - \theta = 0$$

where θ is a threshold that controls the size of the ellipse.

(f) Find a least-squared elliptical fit to the set of points $\{(x_n, y_n)\}$ by finding a value of θ which minimizes the quantity

$$R(\theta) = \sum_{n=1}^{\#E_i} [(\underline{z}_n - \underline{m}_i)' \underline{C}_i^{-1} (\underline{z}_n - \underline{m}_i) - \theta]^2 \qquad (1)$$

It is shown below that both the expected value of (1) and the threshold which minimizes this expression (i.e., gives the optimum fit) are equal to the dimensionality of $\underline{z}$. Thus, given a set of points to be tested for symmetry, we obtain a measure of elliptical symmetry by computing (1) with the optimum threshold and either comparing the result against a perfect ellipse (i.e., zero error in (1)) or against the expected value of (1). The first approach is applicable when a fine measure is desired, while the second approach is more rugged.

A.2 Expected Value of $Q(\underline{z})$

Let

$$Q(\underline{z}) = (\underline{z} - \underline{m})' \underline{C}^{-1} (\underline{z} - \underline{m}) \qquad (2)$$

where $\underline{z}$ is a random vector of dimension d, and $\underline{m}$ and $\underline{C}$ are the mean vector and covariance matrix of the population from which the $\underline{z}$'s are drawn.

Consider the linear transformation

$$\underline{u} = \underline{A}\,\underline{z} \qquad (3)$$

where $\underline{A}$ is a d x d matrix. Then the mean vector of the $\underline{u}$'s is given by

$$\begin{aligned} \underline{m}^* &= E(\underline{u}) \\ &= E(\underline{A}\,\underline{z}) \\ &= \underline{A}\,E(\underline{z}) \\ &= \underline{A}\,\underline{m} \end{aligned} \tag{4}$$

Similarly, the covariance matrix of the $\underline{u}$'s is given by

$$\begin{aligned} \underline{C}^* &= E\{(\underline{u} - \underline{m}^*)(\underline{u} - \underline{m}^*)'\} \\ &= E\{(\underline{A}\,\underline{z} - \underline{A}\,\underline{m})(\underline{A}\,\underline{z} - \underline{A}\,\underline{m})'\} \\ &= \underline{A}\,E\{(\underline{z} - \underline{m})(\underline{z} - \underline{m})'\}A' \\ &= \underline{A}\,\underline{C}\,\underline{A}' \end{aligned} \tag{5}$$

Since $\underline{C}$ is a symmetric matrix, a complete set of orthonormal eigenvectors for this matrix can always be found. If the rows of $\underline{A}$ are chosen as these vectors, then Eq. (3) becomes the Hotelling transform and it is well known that $\underline{C}^*$ will be a diagonal matrix with main diagonal component, λ_k, equal to the variance of the kth component of $\underline{u}$, for k = 1,2,...,d.

From Eqs. (2) through (5),

$$\begin{aligned} Q(\underline{u}) &= (\underline{u} - \underline{m}^*)'\underline{C}^{*-1}(\underline{u} - \underline{m}^*) \\ &= (\underline{z} - \underline{m})'\underline{A}'(\underline{A}')^{-1}\underline{C}^{-1}\underline{A}^{-1}\underline{A}(\underline{z} - \underline{m}) \\ &= (\underline{z} - \underline{m})'\underline{C}^{-1}(\underline{z} - \underline{m}) \\ &= \underline{Q}(\underline{z}) \end{aligned} \tag{6}$$

It then follows that

$$E\{Q(\underline{z})\} = E\{Q(\underline{u})\} \tag{7}$$

However, since $\underline{C}^*$ is a diagonal matrix,

$$E\{Q(\underline{u})\} = E\{(\underline{u} - \underline{m}^*)\ \underline{C}^{*-1}(\underline{u} - \underline{m}^*)\}$$

$$= E\left[\sum_{k=1}^{d} \frac{(u_k - m_k^*)^2}{\lambda_k}\right]$$

$$= \sum_{k=1}^{d} \frac{E\{(u_k - m_k^*)^2\}}{\lambda_k} \tag{8}$$

$$= \sum_{k=1}^{d} \frac{\sigma_k^2}{\lambda_k}$$

where σ_k^2 is the variance of component u_k which, based on the above discussion, is equal to λ_k. From Eqs. (7) and (8), we then have

$$E\{Q(\underline{z})\} = \sum_{k=1}^{d} 1 \tag{9}$$

$$= d$$

That is, the expected value of $Q(\underline{z})$ is equal to the dimensionality of $\underline{z}$.

A.3 Optimum Threshold

Let

$$Q_i(\underline{z}_n) = (\underline{z}_n - \underline{m}_i)^t \underline{C}_i^{-1} (\underline{z}_n - \underline{m}_i) \tag{10}$$

Then (1) may be expressed as

$$R(\theta) = \sum_{n=1}^{\#E_i} [Q_i(\underline{z}_n) - \theta]^2 \tag{11}$$

The minimum of this expression is obtained setting the partial derivative with respect to θ equal to zero and then solving for θ. The result is

$$\hat{\theta} = \frac{1}{\#E_i} \sum_{n=1}^{\#E_i} Q_i(\underline{z}_n) \tag{12}$$

where $\hat{\theta}$ denotes the value of θ which minimizes (11). The right side of Eq. (12) is recognized as an approximation to the expected value of $Q_i(\underline{z})$, where the i in this context denotes the population of vectors belonging to set E_i. It then follows from Eqs. (7), (9), and (12) that

$$\hat{\theta} = E\{Q_i(\underline{z})\} = d \tag{13}$$

In other words, the value of θ which minimizes (11) is equal to the dimension of the $\underline{z}$'s.

For character recognition applications, the $\underline{z}$'s are pixel coordinates and d = 2. Thus, the least square elliptical fit to the points in E_i is obtained in this case by setting $\theta = \hat{\theta} = 2$ in (1).

A.4 Procedure 2

This procedure also uses the eigenvectors described above, but is more general in the sense that it applies to any type of symmetry about two principal axes.

(a) Repeat steps (a) through (d) in Procedure 1.

(b) The perpendicular distance between any point $\underline{z}_n$ in E_i and a line containing $\underline{e}_1$ is given by

$$D_n(i) = \frac{|\underline{z}_n'\underline{e}_2|}{\|\underline{e}_2\|}$$

where $\|\underline{e}_2\| = [c^2 + d^2]^{1/2}$. Compute the average perpendicular distance between this line and all points lying on its positive side ($\underline{z}_n$ lies on the positive side of the line if $\underline{z}_n'\underline{e}_2 > 0$). This average distance is given by

$$D_1^+(i) = \frac{1}{N^+} \sum D_n(i)$$

where the summation is taken over values of n for which $\underline{z}_n'\underline{e}_2 > 0$ and N^+ is the number of points satisfying this condition.

(c) Compute the average perpendicular distance of the points lying on the negative side of the line containing $\underline{e}_1$. This quantity is given by

$$D_1^-(i) = \frac{1}{N^-} \sum D_n(i)$$

where the summation is taken over values of n for which $\underline{z}_n'\underline{e}_2 \leq 0$ and N^- is the number of points satisfying this condition.

(d) Repeat steps (b) and (c) using $\underline{e}_1$ to obtain $D_2^+(i)$ and $D_2^-(i)$.

(e) Define symmetry measures about $\underline{e}_1$ and $\underline{e}_2$ as $s_1(i) = |D_1^+(i) - D_1^-(i)|$ and $s_2(i) = |D_2^+(i) - D_2^-(i)|$ for i = 1,2,...K.

An average measure of deviation from symmetry about the principal axes is given by the respective values of $s_1(i)$ and $s_2(i)$. If, for example, the points in E_i are symmetrical about these axes, then $s_1(i) = s_2(i) = 0$.

APPENDIX B

The following discussion deals with a procedure for corner detection and quantification. The method was developed in an attempt to incorporate both local and global information in the corner detection problem. After experimenting with the technique, however, we found that it lacks sensitivity and that it performs no better than the simpler approaches suggested in our earlier work [1]. The procedure is included here for completeness and because its development contains some concepts that may be useful in other contexts. Its adoption as a useful processing tool is not recommended.

B.1 Background

A corner may be defined as the fortuple $C = (c,\alpha,\beta,\theta)$ where c, the corner point, is the point of intersection of two straight line segments, α and β, with lengths $|\alpha|>0$ and $|\beta|> 0$, respectively. It is assumed that the line segments meet at one of their extremes, forming an interior angle θ. The corner is said to be acute if $0 < \theta < \pi/2$, right if $\theta = \pi/2$, obtuse if $\pi/2 < \theta < \pi$, and degenerate if $\theta = 0$ or $\theta = \pi$.

In the continuous domain, and in the absence of noise, the relationship between θ, $|\alpha|$, and $|\beta|$ becomes important only near degeneracy or when $|\alpha|$ or $|\beta|$ approach zero. In the examples shown in Fig. 1, for instance, one would have difficulty in visually recognizing the presence of a corner only when $\theta\to 0$, $\theta\to\pi$, $|\alpha|\to 0$, or $|\beta|\to 0$. In the presence of noise, however, the relative values of these parameters play a central role in our ability to detect a corner, as illustrated in Fig. 2. Part (a) of this figure shows an acute corner which, for all practical purposes, has been rendered undetectable by the presence of noise. Figure 2(b), by contrast, shows a right corner with the same values of $|\alpha|$ and $|\beta|$ and corrupted by the same amount of noise. This figure is clearly closer to our intuitive concept of a corner, thus illustrating the importance of θ in establishing corner-like properties in a noisy segment.

The relative importance of $|\alpha|$ and $|\beta|$ is illustrated in Fig. 3. Part (a) of this figure shows an undetectable acute corner in which $|\alpha|$ and

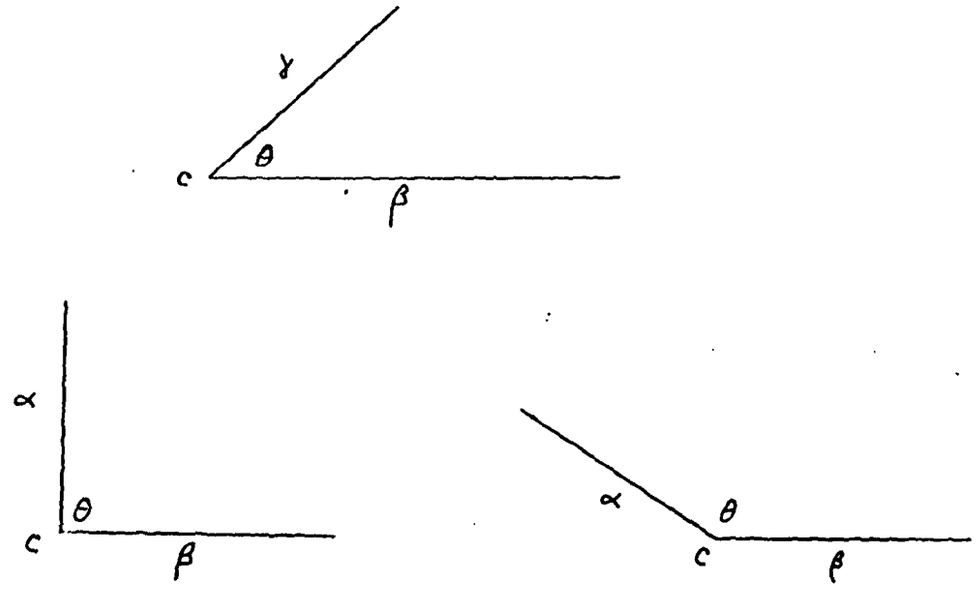


Fig. 1. Examples of acute, right, and obtuse corners.

Fig. 2. Effect of θ in the detectability of corners in noisy segments. (a) Noisy obtuse corner. (b) A right corner with the same values of $|\alpha|$ and $|\beta|$ and corrupted by the same amount of noise.

$|\beta|$ are small with respect to the amount of noise corrupting the segment, while Fig. 3(b) shows a much more corner-like segment; this figure has the same amount of noise, but much larger values of $|\alpha|$ and $|\beta|$. This effect is clearly analogous to the concept of increased signal to noise ratio in communication theory, in the sense that the greater the corruption the larger $|\alpha|$ and $|\beta|$ would have to be to define a corner-like segment.

If we view the process of digitizing a segment as a mechanism that distorts (i.e., introduces noise to) the spatial integrity of the segment, it is evident from the preceding discussion that the relationship between the amount of distortion and the parameters θ, $|\alpha|$, and $|\beta|$ is an essential consideration in the development of any procedure for detecting and evaluating corners in digital segments. The examples given in Figs. 2 and 3 also illustrate the futility of using local corner detectors or curve-tracing techniques which do not take into account the values of these or similar parameters for finding corners in digital segments.

B.2 Corner Detection and Evaluation

The procedure developed in this section is based on the idea of utilizing a model of an ideal corner in order to establish a measure of corner "quality" which takes into account segment distortion and the parameters θ, $|\alpha|$, and $|\beta|$ defined in the previous section. The following discussion applies only to simple, thinned digital segments (i.e., thinned segments which do not cross themselves) with only two distinct end points. Multiply-connected segments can be handled by decomposition into simple segments at the branch points.

With reference to Fig. 4, let $C = (c,\alpha,\beta,\theta)$ denote an ideal (noiseless) continuous corner withend points a and b, and denote by λ a bound on the spatial distortion of α and β as a result of digitizing C. The parameter λ could be, for example, a function of the variance of the points in a digital segment referenced to the straight line segments α and β.

In order to relate C and the "broad" corner defined by the region between the dashed boundary in Fig. 4, it is necessary to establish a proportionality factor that involves λ, α, β, and θ. This can be accomplished with the aid of Fig. 5. Let d be a straight line segment

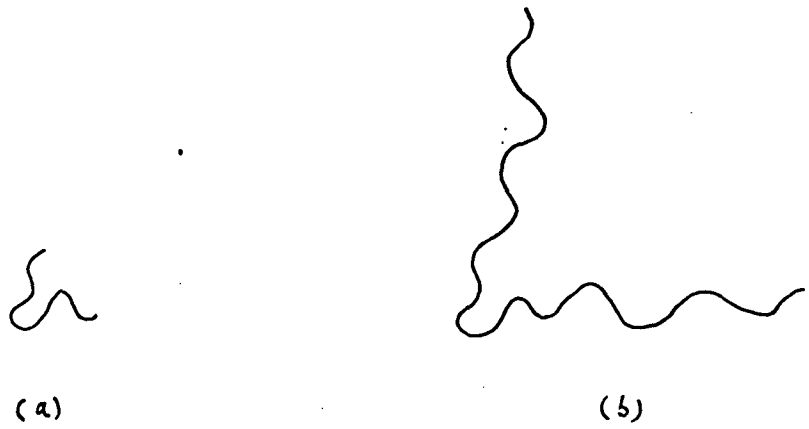


Fig. 3. Effect of $|\alpha|$ and $|\beta|$ in the detectability of corners in noisy segments. (a) Noisy acute corner. (b) A corner with the same angle and noise, but with larger values of $|\alpha|$ and $|\beta|$.

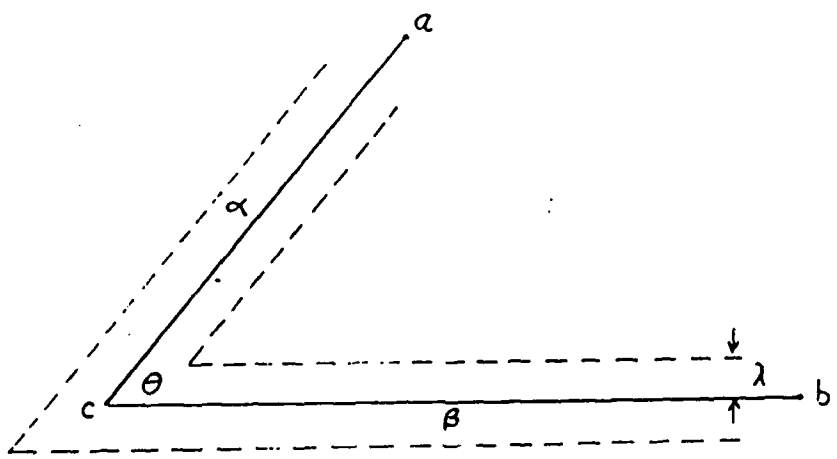


Fig. 4. An ideal corner and bound on the spatial distortion of the segments α and β.

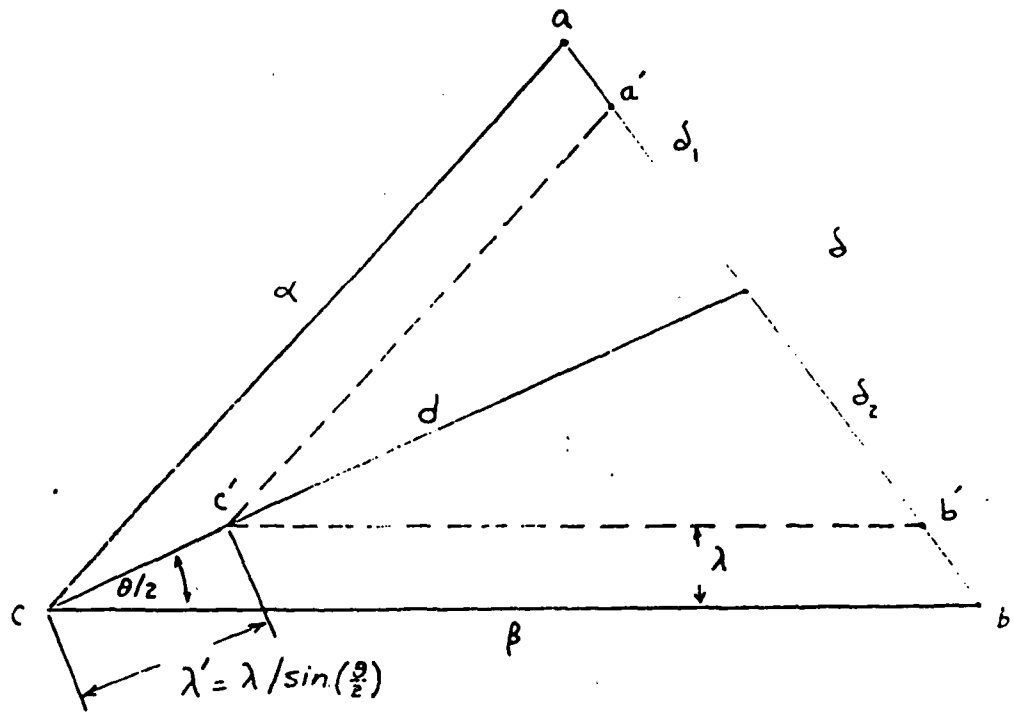


Fig. 5. Geometrical arrangement used to derive a proportionality factor between an ideal and corresponding distorted corner.

starting at c, bisecting θ, and ending at the intersection of d with line δ. The projection of λ along d is given by

$$\lambda' = \frac{\lambda}{\sin(\frac{\theta}{2})} \tag{1}$$

As λ increases for a fixed value of θ, λ' increases and the area of triangle a'c'b' decreases proportionally. Similarly, for fixed λ, a decrease in θ causes λ' to increase and the area of a'c'b' to decrease. Clearly, the smaller this area, the greater the difference between the ideal corner and a distorted corner with bound λ. Suppose, however, that we require that the distorted corner be scaled so that the area a'c'b' is equal to the area associated with the ideal corner (i.e., the area of abc). From elementary trigonometry, it then follows that the length of line d must be extended to $|d_e| = |d| + \lambda'$. Writing this as a proportion, we have

$$\frac{|d_e|}{|d|} = 1 + \frac{\lambda'}{|d|} \tag{2}$$

or, using Eq. (1):

$$\gamma = 1 + \frac{\lambda}{|d|\sin(\frac{\theta}{2})} \tag{3}$$

where γ is the proportionality factor $|d_e|/|d|$.

By using the law of sines, it follows from Fig. 5 that $\frac{|\delta_1|}{\sin(\frac{\theta}{2})} = \frac{|d|}{\sin\theta_a}$ and $\frac{\sin\theta}{|\delta|} = \frac{\sin\theta_a}{|\beta|}$ so that

$$|d| = \frac{|\delta_1||\beta|\sin\theta}{|\delta|\sin(\frac{\theta}{2})} \tag{4}$$

Since d bisects θ, we also have the relation $\frac{|\delta_1|}{|\delta_2|} = \frac{|\alpha|}{|\beta|}$ and, using the

fact that $|\delta_1| + |\delta_2| = |\delta|$, it follows from Eq. (4) that

$$|d| = \frac{|\alpha||\beta|\sin\theta}{(|\alpha|+|\beta|)\sin\frac{\theta}{2}} \tag{5}$$

Substitution of Eq. (5) into Eq. (3) yields

$$\gamma = 1 + \frac{(|\alpha| + |\beta|)\lambda}{|\alpha||\beta|\sin\theta} \tag{6}$$

This equation may be expressed in terms of the shorter of the two segments by letting

$$L_{min} = \min(|\alpha|,|\beta|) \tag{7}$$

and

$$r = \frac{\max(|\alpha|,|\beta|)}{\min(|\alpha|,|\beta|)} \tag{8}$$

Substitution of Eqs. (7) and (8) into Eq. (6) yields

$$\gamma = 1 + \frac{(1 + r)\lambda}{rL_{min}\sin\theta} \tag{9}$$

DISTRIBUTION LIST

Department of the Navy
Asst Secretary of the Navy
(Research Engineering &
System)
Washington DC 20350 (1)

Project Manager
ASW Systems Project (PM-4)
Department of the Navy
Washington DC 20360 (1)

Department of the Navy
Chief of Naval Material
Washington DC 20360 (1)

Department of the Navy
Chief of Naval Operations
ATTN: OP 951
Washington DC 20350 (1)

Department of the Navy
Chief of Naval Operations
ATTN: OP 952
Washington DC 20350 (1)

Department of the Navy
Chief of Naval Operations
ATTN: OP 980
Washington DC 20350 (1)

Director
Defense Technology Info
Cen
Cameron Station
Alexandria, VA 22314 (12)

Department of the Navy
Director of Navy
Laboratories
Rm 1062 Crystal Plaza
Bldg 5
Washington DC 20360 (1)

Commander
DWTaylor Naval Ship
R & D Cen
Bethesda MD 20084 (1)

Commanding Officer
Fleet Numerical Ocean Cen
Monterey CA 93940 (1)

Commander
Naval Air Development Cen
Warminster PA 18974 (1)

Commander
Naval Air Systems Command
Headquarters
Washington DC 20361 (1)

Commanding Officer
Naval Coastal Systems Cen
Panama City FL 32407 (1)

Commander
Naval Electronic Sys Com
Headquarters
Washington DC 20360 (1)

Commanding Officer
Naval Environmental
Prediction
Research Facility
Monterey CA 93940 (1)

Commander
Naval Facilities Eng Com
Headquarters
200 Stovall St.
Alexandria VA 22332 (1)

Commanding Officer
Naval Oceanographic Office
NSTL Station
Bay St. Louis, MS 39522 (1)

Commander
Naval Oceanography Command
NSTL Station MS 39522
Bay St. Louis, MS 39522 (1)

Director, Liaison Office
Naval Ocean R&D Activity
800 N. Quincy Street
502 Ballston Tower #1
Arlington VA 22217 (1)

Commander
Naval Ocean Systems Center
San Diego CA 92152 (1)

Superintendent
Naval Postgraduate School
Monterey CA 93940 (1)

Commanding Officer
Naval Research Laboratory
Washington DC 20375 (1)

Commander
Naval Sea System Command
Headquarters
Washington DC 20362 (1)

Commander
Naval Surface Weapons Cen
Dahlgren VA 22448 (1)

Commanding Officer
Naval Underwater Systems
Cen
ATTN: New London Lab
Newport RI 02840 (1)

Department of the Navy
Office of Naval Research
ATTN: Code 102
800 N. Quincy St.
Arlington VA 22217 (1)

Officer in Charge
Office of Naval Research
Detachment, Boston
Barnes Building
495 Summer St.
Boston, MA 02210 (1)

Commanding Officer
ONR Branch Office LONDON
Box 39
FPO New York 09510 (1)

Commanding Officer
ONR Western Regional Ofs
1030 E. Green Street
Pasadena CA 91106 (1)

Director
Scripps Inst of Oceanography
Univ of Southern California
La Jolla CA 92093 (1)

Working Collection
Texas A & M University
Department of Oceanography
College Station, TX 77843 (1)

President
Woods Hole Oceanographic Inst
Woods Hole, MA 20543 (1)

Director
Defense Mapping Agency
Washington, DC 20305 (5)

Director
Defense Mapping Agency
Hydrographic/Topographic Cen
6500 Brooke Lane
Washington, DC 20315 (8)

Director
Defense Mapping Agency
Aerospace Cen
St. Louis Air Force Station,
MO 63118 (5)

Director
Defense Mapping AGency
Special Program Office of
Exploitation and
Modernization
8301 Greensboro Drive,
Suite 1100
McLean, VA 22102 (2)

Commanding Officer
Naval Ocean R & D Activity
NSTL Station, MS 39529 (4)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NORDA Technical Note 185	2. GOVT ACCESSION NO. AD-A129 658	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Syntactic/Semantic Techniques for Feature Description and Character Recognition		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. C. Gonzalez		8. CONTRACT OR GRANT NUMBER(s) P. O. N00014-80-M-0030
9. PERFORMING ORGANIZATION NAME AND ADDRESS Perceptics Corporation 221B Clark Street Knoxville, Tennessee 37921		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS P.E. 63701B
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Ocean Research & Development Activity Pattern Analysis Branch NSTL Station, Mississippi 39529 ATTN: R.M. Brown		12. REPORT DATE January 1983 13. NUMBER OF PAGES 70
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as Item 11		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Character recognition	Attributes	Grammar	Interactive
Alphanumeric	Features	Strings	
Parameter	Hierarchical	Syntactic	
Automaton	Branch points	Semantic	
Primitives	Polygonal	Structural	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A unified approach for the development of a syntactic/semantic character recognition system is discussed. The system is based on a hierarchical, semantics-based feature extractor that produces coded strings suitable for processing with finite automata whose transitions are governed by both the structure and semantics of the input patterns. A procedure is given for checking class separability, and an interactive approach is proposed as a tool for efficient system design using training pattern classes.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


DATE
ILME